# Universidad Nacional Autónoma de México

## FACULTAD DE INGENIERÍA

## “SISTEMA DE MONITOREO DE SERVIDORES UNIX”

# T E S I S

Que para obtener el Título de:

INGENIERO EN COMPUTACIÓN

P r e s e n t a:

EDGAR CRISTÓBAL RAMÍREZ MIRANDA


Director de Tesis: Ing. Noé Cruz Marín

Agradecimientos

Más que gracias, mi eterno amor a mi padre, mi madre y a mis hermanos a los cuales les debo lo que soy.

Papá te admiro y respeto profundamente por la rectitud y honradez con la que te conduces a cada paso de tu vida, por el apoyo y dirección que siempre de ti he recibido, por todas las enseñanzas que desde chico recibí de ti y espero seguir recibiendo.

Mamá te amo por todos los años de dedicación que me has dado, por los desvelos, por las lágrimas, por los cariños, por tu inmensa ternura y compasión, por que no encuentro ni encontraré jamás la forma de pagarte todo lo que me has dado, gracias.

A mi hermano Emilio por ser mi amigo, por enseñarme a defenderme, por ser mi ídolo, por ensañarme a no desistir nunca hasta alcanzar las metas, porque a tu lado siempre me sentí protegido, por alentarme a ser ingeniero.

A mi hermano Joaquín, por que de ti aprendí a ser más sereno a pensar antes de actuar, por siempre protegerme de todos sin importarte nada, por estar conmigo en las buenas y en las malas, por tu tolerancia, por ser mi amigo en los tiempos difíciles.

Gracias por aguantarme, soportarme, cuidarme y quererme de la forma en la que lo hacen, estoy y siempre estaré orgulloso de pertenecer a la familia Ramírez Miranda.

Edgar Cristóbal Ramírez Miranda

A la Universidad Nacional Autónoma de México, por permitirme el honor de formar parte de ella. Trataré siempre de representarla dignamente.

A la Facultad de Ingeniería por ser mi casa y proporcionarme las herramientas necesarias para enfrentarme al mundo. De igual forma haré siempre mi mejor esfuerzo por representarla dignamente.

A todas las personas que han compartido parte de su vida conmigo y que con ello han enriquecido la mía, a mis amigos por hacer más llevadera la vida y por ser los hermanos que uno escoge en esta vida, a mis profesores por compartir su conocimiento, su modo de ver y apreciar la vida y resolver los problemas que en ella se presentan.

A mi abuelo paterno a quien recuerdo con mucho cariño, a mi abuelita materna a quien quise y siempre querré, a todos mis tíos, tías, primos, primas, sobrinos y sobrinas que por ser tantos no terminaría de nombrar pero que estimo en gran medida.

A mis compañeros de trabajo que se convierten de manera involuntaria en familia temporal y compañía diaria.

Edgar Cristóbal Ramírez Miranda

**SISTEMA DE MONITOREO DE SERVIDORES UNIX**

## Índice


# Prefacio

## Objetivos

El trabajo aquí descrito y sustentado está encaminado a cumplir con los siguientes objetivos:

- Diseñar, desarrollar e implementar un sistema de monitoreo configurable que permita obtener información valiosa de los servidores y de los procesos que sobre ellos se ejecutan, para garantizar la operación óptima y en su caso actuar con oportunidad con acciones correctivas.

- Implementar un sistema de monitoreo que permita garantizar niveles altos de seguridad en cuanto la manera de obtener, transferir y desplegar la información que de los servidores se obtenga.

- Brindar un entorno amigable e intuitivo para la administración y operación del sistema de monitoreo.

## Introducción

En la actualidad la sistematización de procesos y tareas de toda índole es una tendencia en las organizaciones de todos tamaños. Los procesos productivos realizados de manera manual son plasmados y sustituidos por sistemas de cómputo que permiten el ahorro de recursos, tiempo y costos, hacen más eficientes y transparentes las acciones realizadas, permiten la comunicación con otros sistemas, comparten y generan información y forman parte importante en la toma de decisiones proporcionando datos más sólidos.

Dichos sistemas pasan de ser innovaciones, a sistemas comunes y después a herramientas necesarias y hasta indispensables para la realización de las actividades cotidianas. La dependencia que se genera hacia dichos sistemas (poca o mucha), trae consigo la necesidad de garantizar que los servicios, recursos o facilidades que éstos proporcionan, estén disponibles cuando sean requeridos y proporcionen las respuestas esperadas, es decir, presenten un comportamiento en tiempo y forma apegados a su diseño.

En general dichas aplicaciones no funcionan por si solas ni con recursos propios, éstas, necesitan de entes externos que les proporcionen los recursos de almacenamiento, conectividad, procesamiento, memoria etcétera, indispensables para la ejecución de las tareas y funciones programadas en las aplicaciones. Estos entes denominados servidores son en la mayoría de los casos dispositivos de cómputo con capacidades suficientes para operar por largos periodos de tiempo y proporcionar los recursos necesarios para la ejecución de las aplicaciones.

La revisión periódica de la condición o estado de salud de los servidores es importante y necesaria para garantizar la continuidad y buen funcionamiento de las aplicaciones y servicios que a través de ellos se proporcionan y representa una de las tareas principales de los administradores de sistemas computacionales. Dicha revisión se denomina monitoreo, el cual puede ser activo o pasivo. Sin importar el tipo de monitoreo del que se trate éste intentará determinar el estado de salud del servidor a través de la lectura y análisis de los valores de parámetros definidos, los cuales deberán ser representativos y acordes al tipo y objeto de monitoreo de que se trate.

El método para la extracción de los datos del monitoreo puede ser manual o automatizado, es decir, por medio de la ejecución, manual de comandos o lectura e interpretación de bitácoras o a través de alguna herramienta de monitoreo que automatice la extracción de la información. La diferencia principal de un método u otro radica en la rapidez con la que uno u otro realizan el trabajo, dado que en general no existe tarea alguna que un buen administrador no pueda realizar de la misma forma que una herramienta automatizada.

La seguridad de equipos de cómputo en entornos de redes (ya sea entorno LAN, WAN o Internet) es hoy en día un elemento de gran importancia para las empresas y/o instituciones. Los ataques a servidores conectados a redes sigue incrementándose año tras año y con ello la complejidad de los métodos empleados por los atacantes.

El caso del monitoreo automatizado a través de sistemas de cómputo no es la excepción. La información que de los servidores se extrae puede tener diferentes valores (dependiendo del entorno, el tipo de información que en ellos se maneje etc.). El monitoreo es una herramienta que proporciona información del estado y perfil de los servidores, esta información también debe ser protegida ya que dicha información también podría ser valiosa para posibles atacantes.

Ya sea que la información contenida en los sistemas pueda ser de carácter confidencial, privado, secreto o público, aun así es necesario garantizar que esta continuará estando de la forma que se espera, con la disponibilidad esperada y accesible sólo a aquellos a los que se pretende este dirigida.

Hoy en día las áreas de tecnologías son claves para el soporte de los procesos ya sean estos procesos operativos, administrativos o de negocios.

La observación del comportamiento de los sistemas y de los servicios que estos proporcionan para garantizar que estos se mantengan en valores óptimos o aceptables para así garantizar la continuidad de los procesos que sustentan la operación de la organización o institución, es el principal objetivo del monitoreo.

Más aún, las lecturas y análisis de datos de un monitoreo, permiten establecer tendencias, así como, anticipar eventualidades.

**La importancia de detectar fallas o errores oportunamente**

En la actualidad muchos de las actividades que realizamos están sistematizadas, es decir, los procesos y procedimientos que ellas involucran fueron abstraídas y plasmadas en flujos de datos con entradas, operaciones y salidas o resultados y finalmente introducidas en sistemas de cómputo montados en servidores.

Los servidores proporcionan los recursos por medio de los cuales los sistemas procesan y transforman los datos de entrada en información valiosa y útil para los usuarios. Con el paso del tiempo nos adaptamos más y más a los servicios y beneficios que los sistemas de cómputo nos proporcionan y la falla o suspensión de estos servicios puede ocasionar molestias, grandes problemas o pérdidas monetarias o de oportunidades.

Por lo explicado anteriormente es importante detectar posibles fallas o errores en los servidores de manera anticipada y oportuna de tal manera que se pueda evitar la discontinuidad en la operación de los servicios proporcionados o en caso de

presentarse algún evento de interrupción del servicio, nos ayude a reducir el tiempo de respuesta y el impacto nocivo de esta discontinuidad en la operación.

**Ayuda en la toma decisiones para el adecuado uso de los recursos**

Los datos que por medio de un esquema de monitoreo se obtienen, no se despliegan tal y como se obtienen. En muchos de los casos eso no tendría valor alguno, es el análisis posterior de éstos, lo que les da sentido o significado.

Para lograr extraer información de los datos, es necesario someterlos a procesos de clasificación, discriminación, transformación, cuantificación y calificación, con el propósito de desplegar sólo la información sustancial. La calificación de los datos depende de parámetros de medición hasta cierto punto subjetivos los cuales nos indican la calidad o estado de lo medido. En general se habla de valores base o línea de base, los cuales definen el estado ideal de un sistema, así mismo se establecen valores intermedios a los cuales se les define como valores de alerta y valores tope o de cota superior que indican los valores máximos permitidos y que generalmente se asocian a valores de alarmas.

La información es la materia prima para la toma decisiones, el cual es un proceso de selección entre dos más alternativas posibles, las decisiones son el motor de los negocios o de la operación. De la adecuada selección de las alternativas depende en gran parte el éxito o fracaso de cualquier organización.

Los administradores consideran a veces la toma de decisiones como su trabajo principal, dado que constantemente tienen que decidir, ¿qué debe o no hacerse?, ¿quién ha de hacerlo?, ¿cuándo y dónde?, y en ocasiones hasta ¿cómo se hará?.

El conjunto de toma de decisiones conforma la planeación, la cual define el modo, la forma y  la dirección hacia la que una organización desea caminar.

La información que el monitoreo proporciona es una herramienta base para la toma de decisiones en cuanto a recursos computacionales, dado que gracias a él, es posible detectar deficiencias en los servidores, sobrecarga de procesos, cambio de patrones de uso u obsolescencias de los recursos. Para los administradores, el proceso de toma de decisión es sin duda una de las mayores responsabilidades y el contar con herramientas que les proporcione información oportuna y confiables es sin duda una necesidad primaria.

**Alcances del proyecto de tesis**

El proyecto pretende proporcionar información valiosa, clara, oportuna y confiable de manera tal que ésta, no pueda ser leída o modificada por terceros sin autorización explicita.

Los métodos que el sistema de monitoreo utiliza no son intrusivos, con el propósito de no generar falsos positivos en el análisis de ninguna otra herramienta de monitoreo de red.

Se utiliza un canal convencional de administración (puerto 22 de tcp con ssh) con el propósito de no abrir más puntos de contacto en el servidor que podrían reducir su nivel de seguridad.

El sistema no pretende proporcionar información del comportamiento de la red, debido a que esta tarea es responsabilidad de los administradores de red, no de los administradores de servidores aun y cuando en muchos casos sea el mismo personaje desarrolle ambas actividades.

El proyecto pretende servir de herramienta para una de las actividades principales de los administradores, es decir, el monitoreo del desempeño y salud de los servidores sobre los cuales se sustentan los servicios de operación de las organizaciones o instituciones, de tal forma que se garantice la continuidad de la operación.

**Información y datos útiles que el monitoreo proporciona**

La herramienta de monitoreo verifica tres aspectos de los servidores, la comunicación, los servicios que estos proporcionan y su estado de salud.

En la parte de comunicación el monitoreo nos informa si el servidor objeto de monitoreo está activo, es decir, si está conectado a la red, encendido y puede comunicarse.

En la parte de servicios que el servidor proporciona, nos indica si los puertos por los que estos servicios se proporcionan están abiertos y escuchando, además de verificar si los procesos motivo de monitoreo están ejecutándose.

En la parte de revisión de la salud de los servidores, el monitoreo verifica parámetros de memoria RAM y SWAP, utilización de procesador, utilización de los sistemas de archivos, usuarios conectados, parámetros de red generales entre otros.

Adicionalmente el sistema de monitoreo informa si se ha abierto un nuevo puerto en el servidor, lo cual podría significar un evento de intrusión, debido a que en general las aplicaciones no abren puertos de manera a aleatoria ni intempestiva.

**Mejoras que un esquema de monitoreo proporciona**

El monitoreo analizado desde el punto de vista de herramienta reactiva, permite reducir los tiempos de respuesta ante eventualidades, dado que indica el problema de forma aislada y gracias a ellos es posible atacar al problema de manera directa y así encontrar más rápido la solución.

El monitoreo analizado desde el punto de vista de herramienta proactiva, permite identificar fallas por sobre carga en la memoria RAM o SWAP, espacio disponible en sistemas de archivos, sobre carga de CPU antes de que estos alcancen niveles que impidan la continuidad de la operación.

El monitoreo reduce los costos ocasionados por interrupciones en el servicio.

El monitoreo permite a los administradores tener la certeza de que los servidores están en condiciones adecuadas para seguir proporcionando los recursos de manera adecuada tales que soporten el funcionamiento de los servicios.

El monitoreo permite que el tiempo del o de los administradores u operadores se utilice de forma más eficiente y que estos tengan la posibilidad de utilizar su ingenio en asuntos más creativos.

## Entorno de trabajo

Esta herramienta de monitoreo esta desarrollada para trabajar sobre la mayoría de los ambientes UNIX dado que el motor de extracción de datos esta basado sólo en comandos genéricos de interprete de comandos, a su vez puede trabajar sobre cualquier tipo de redes TCP/IP dado que el único requerimiento para que ésta opere es que el puerto 22 de tcp este habilitado y ejecutando sshd en él. La extracción de datos se realiza sobre el mismo canal de administración por lo que no se considera una herramienta intrusiva, a su vez todo el tráfico de datos entre el cliente y el servidor transita de forma cifrada por lo que es confiable aun en entornos públicos no seguros como el Internet.

## Resumen capitular

En el primer capítulo de este proyecto de tesis se abordan diferentes metodologías de ingeniería del software para el análisis, diseño, desarrollo e implementación de aplicaciones computacionales, las cuales proporcionan guía y marco teórico para la implementación del sistema de monitoreo de servidores Unix.

Estas metodologías presentan diferentes enfoques para la resolución de los problemas, cada una de ellas con pros y contras, y en general no aplicables para todos los casos, pero sin embargo aportan los lineamientos base para formar un enfoque sistemático en la resolución de cualquier problema de cómputo.

El segundo capítulo presenta el marco histórico, tanto del entorno de red representado por el conjunto de protocolos TCP/IP, protocolos orientados a proporcionar servicios de monitoreo y administración de dispositivos de red como el protocolo SNMP en sus diferentes versiones, así como del entorno del sistema operativo Unix. En este capítulo también se evalúan herramientas, lenguajes de programación, motores de bases y el servidor de http, todos ellos utilizados para el desarrollo del sistema de monitoreo. Por último también se aborda la necesidad de darle al sistema de monitoreo un enfoque orientado a la seguridad, por lo que se presentan algunos conceptos básicos de seguridad, cifrado y esquemas de comunicación segura, mismos que se implementan en el proyecto de tesis.

El capítulo tres expone el concepto de monitoreo y, define y diferencia los tipos de monitoreos y sus características particulares. Presenta herramientas de monitoreo, cada una de ellas orientadas necesidades particulares de monitoreo y realiza un comparativo entre estas opciones de solución de monitoreo.

El cuarto y último capítulo expone el análisis y desarrollo sistematizado que se realizó para la solución y cumplimiento de los requerimientos del sistema de monitoreo, el enfoque de seguridad aplicado al desarrollo del proyecto de tesis y las herramientas que tras el análisis y comparación se utilizaron para el desarrollo de dicho sistema de monitoreo, así como los resultados obtenidos tras su implementación.


# Capítulo I

# La ingeniería y el software

## 1.1 Proceso de diseño

Se puede decir que todos o casi todos los problemas o situaciones cuentan con diferentes soluciones, el tipo de solución y la manera de llegar a ella es el que hace de uno u otro método el más adecuado para una u otra situación. La ingeniería proporciona procedimientos generales que pueden ser empleados como guía para la obtención de soluciones funcionales y útiles.

El diseño es una de las primeras etapas en el proceso de generación de una solución, en la cual se define el problema y plantea una posible solución de manera general.

Edward V. Krick en “Introducción a la ingeniería y al diseño en la ingeniería” expone un procedimiento general para resolver problemas de ingeniería, el cual se presenta en las siguientes cinco fases.

- *Formulación del problema*: el problema de que se trate se define en forma amplia y sin detalles.
- *Análisis del problema*: en está etapa se le define con todo detalle.
- *Búsqueda de soluciones*: las soluciones alternativas se reúnen mediante indagación, investigación, invención, etcétera.
- *Decisión*: todas las alternativas se evalúan, comparan y se seleccionan hasta que se obtiene la solución óptima.
- *Especificación*: la solución elegida se expone por escrito detalladamente.

Este procedimiento de cinco pasos es aplicable a cualquier cosa que un ingeniero diseñe.

Formulación del problema

La definición del problema de manera clara y completa es la primera tarea que el diseño debe atender, sin la definición del problema es virtualmente imposible alcanzar la solución esperada, es decir, no se puede resolver un problema sin saber en qué consiste éste.

Es conveniente tener una vista panorámica del problema desde el principio, porque una vez que uno se sumerge en los detalles es materialmente imposible tener una amplia perspectiva. Por lo tanto los objetivos principales de la formulación de un problema son definir en términos generales en que consiste y cuales son las entradas y salidas esperadas.

La formulación o enunciado del problema debe ser lo más amplia posible y esta debe de tender hacia una definición menos concreta de los estados inicial y final (o entradas y salidas), los cuales podemos definir como A y B respectivamente y no concentrarse en los estados intermedios, ya que estos surgirán en dependencia con la o las soluciones propuestas. Al plantear el problema en estos términos es posible incluir un mayor número de posibles soluciones al problema.

En general, debe tratarse siempre de formular un problema de modo que comprenda o incluya tanto del problema total como sea posible.

La formulación del enunciado del problema puede tocar elementos de decisión o de responsabilidad de otras personas o áreas y en dicho caso se tendrá que reconciliar dichos conflictos de intereses o reformular el enunciado del problema de tal forma que no invada dicho espacio.

El grado en que se justifique y pueda uno llevar a cabo la formulación amplia de un problema dependerá del alcance de nuestras responsabilidades o autoridad, de la importancia del problema y de la limitación o limitaciones (si las hay) de tiempo y dinero que se tenga para la resolución del problema.

La "entrada" que interviene en la fase de formulación es una información vaga y mezclada con hechos sin importancia y confusos, acerca de lo que se necesita o se quiere. La "salida", una provechosa formulación del problema, se convierte en "entrada" para la siguiente fase del proceso de diseño, el análisis del problema.

Análisis del problema

En la formulación del problema es suficiente identificar los estados A y B, sin embargo, para resolver el problema es necesario saber más acerca de la entrada y la salida (A y B). Por lo tanto, durante esta etapa del proceso de diseño se determinan las características cualitativas y cuantitativas de los estados A y B.

La caracterización de los estados A y B se compone elementos estáticos y/o dinámicos a estos elementos les conoce como variables de entrada y variables de salida, respectivamente. Generalmente existen límites para el valor en que pueden fluctuar tales variables. Para que un ingeniero pueda resolver satisfactoriamente un problema, deberá contar con estimaciones de confianza de los valores de las variables y limitaciones de entrada y de salida.

Otro elemento que interviene en la fase de análisis del problema son las restricciones, una restricción es una característica de una solución que se fija previamente por una decisión, por la naturaleza del problema, por requisitos legales o normativos o por cualquier otra disposición que se tenga que cumplir.

Estas restricciones limitan las alternativas que se pueden proponer para la solución del problema. Sin embargo algunas de las restricciones provienen de decisiones sin fundamento o sin el correcto análisis o por mera suposición, que el ingeniero deberá detectar y exponer con el fin de ampliar el número de soluciones posibles, a este tipo de restricciones se le denomina restricciones ficticias.

Es aconsejable que el ingeniero mantenga una perspectiva abierta (de vaso medio lleno y no de vaso medio vacío) y que entienda que el objetivo del análisis no es conocer todas las formas de restricción, sino darse cuenta de cuáles son las formas en que hay restricción alguna, y posteriormente aprovechar esta libertad en la búsqueda de soluciones.

El definir correctamente los valores y variables de entrada permitirá diseñar una solución capaz de responder a las entradas reales, a su vez el definir correctamente los valores y variables de salida permitirá diseñar las funciones y algoritmos necesarios para transformar los datos de entrada en las salidas reales esperadas.

Los criterios que se utilizan para seleccionar el mejor diseño deben identificarse durante el análisis del problema. Realmente, los criterios cambian muy poco de problema en problema; el costo de construcción o fabricación, la seguridad, la confiabilidad, la facilidad de mantenimiento, la funcionalidad, entre otros. Lo que si cambia significativamente de problema en problema, es la importancia relativa de criterios en particular. De ahí que respecto a los criterios la tarea principal sea ordenar la importancia relativa de éstos.

Un criterio especialmente importante afectará a los tipos de soluciones que se destacan en la búsqueda de alternativas, y este hecho debe ser conocido antes que principie tal búsqueda.

Otro elemento a tomar en cuenta dentro de la búsqueda y análisis de la solución es la utilización, está determina en primera instancia el número de veces que se utilizará la solución, la frecuencia de uso o la particularidad o generalidad de casos que la solución atienda. La utilización esta relacionada en la mayoría de los casos a los costos de desarrollo, fabricación y producción de la solución, dado que si por ejemplo un dispositivo solo se reproducirá 10 veces y una de sus piezas en muy cara, esto no afectará realmente, dado que la intención es construir el dispositivo por especial que este sea, por otro lado si se planea fabricar en masa dicho dispositivo habrá que plantear la sustitución de la pieza cara por una similar pero mucho más barata.

El análisis del problema comprende mucho trabajo de reunión y procesamiento de información. El resultado es una definición del problema en detalle, la cual maximiza las probabilidades de hallar una solución óptima. Ahora ya se está listo para iniciar la búsqueda de tal solución.

Búsqueda de soluciones

En esta fase del proyecto de diseño se buscan activamente las soluciones posibles y uno se lanza a lo que es una verdadera búsqueda o investigación, en la mente, en la literatura técnica y científica y en el mundo que nos rodea. La vasta acumulación de conocimientos humanos proporciona soluciones “ya hechas” para algunas partes de los problemas. El buscar tales soluciones es un proceso relativamente directo, que consiste en explorar nuestra memoria, consultar libros, informes técnicos, y aplicar prácticas existentes, así como hacer uso del proceso mental llamado invención.

La cantidad de conocimiento que se tenga de las ramas involucradas en el problema, posibilitará el encuentro de una mejor solución y de una manera más rápida.

La cantidad de conocimientos y habilidades es un factor de peso en la búsqueda y evaluación de posibles soluciones, debido a que muchos de estos conocimientos acumulados nos ayudaran, tanto a encontrar partes de la solución, como a descartar con mayor certeza soluciones muy costosas o no viables. La inventiva, la imaginación y la creatividad son un factor muy importante también, ya que significan la proposición de soluciones diferentes a las ya establecidas o el motor para mejorarlas.

El conocimiento no es un recurso estático, este tal vez tenga que ser ampliado, refinado o mezclado con otros elementos de conocimiento para lograr encontrar soluciones a partes del problema.

Conforme se avanza en la búsqueda de la solución es muy probable que se deba desglosar el problema en partes del mismo (pero sin llegar al detalle puntual) y atender cada una por separado, ya sea que se atienda una a la vez o al mismo tiempo cuando estas no están directamente ligadas.

Finalmente, muchas alternativas pueden evaluarse satisfactoriamente mientras se hallan todavía en un estado de especificación relativamente burdo; puesto que la mayoría de ellas serán rechazadas, ¿por qué desperdiciar tiempo en sus detalles?

De hecho es mejor formar sólo conceptos de solución en esta fase del proceso de diseño. (Un concepto de solución es la esencia, el espíritu o la naturaleza general de

una solución particular, su forma puede ser un croquis, unas cuantas palabras, una frase o dos).

La búsqueda de la solución óptima es un proceso que puede continuar de manera indefinida, pero claro esta, que en la vida real siempre existen límites, como por ejemplo límites temporales, económicos o ambos, etcétera. Es difícil definir si la solución que se encontró, es la óptima con respecto a los límites de búsqueda. Lo que si es posible establecer respecto a la elección de la solución, es que esta decisión debe tomarse antes de llegar a un valor de utilidad decreciente.

Para ilustrar los extremos y ayudar a aclarar este punto, se propone la siguiente cuestión: ¿Qué sería preferible, tener más alternativas de donde escoger pero menos tiempo para evaluarlas, a riesgo de no poder seleccionar la mejor, o bien dedicar más tiempo a evaluar exhaustivamente menos soluciones y, por lo tanto, estar seguros de haber elegido la mejor de la muestra, pero no haber encontrado la mejor?.

Decisión

En la fase de búsqueda se amplía el número y la variedad de las soluciones posibles. En la fase de decisión lo que se necesita es un procedimiento de eliminación que reduzca estas alterativas a la solución preferible.

Inicialmente, las soluciones elegibles se expresan sólo en términos generales, quizá con palabras o bosquejos. Después de que hayan sido eliminadas las alternativas obviamente deficientes o de inferior calidad, con frecuencia por procedimientos de evaluación relativamente rápidos y burdos, se añaden más detalles a las posibilidades restantes, las que se evaluarán mediante métodos más refinados. Este proceso de depuración en varias etapas continuará hasta que surja la solución preferible. A medida que se avanza se evalúan diferentes combinaciones de soluciones parciales para determinar la óptima.

Aunque los aspectos específicos varían de un caso a otro, en casi todo problema hay que dar los cuatro pasos siguientes antes de que pueda llegarse a una inteligente decisión de diseño:

1. Seleccionar los criterios y determinar su importancia relativa.
2. Predecir el funcionamiento de las soluciones alternativas con respecto a tales criterios.
3. Comparar las alternativas sobre la base de los funcionamientos predichos.
4. Hacer la elección.

El proceso de decisión varía desde los procedimientos exhaustivos más elaborados que comprenden medición, investigación, predicción y comparación de costos en alto grado, hasta el simple juicio informal y rápido.

Criterios como la confiabilidad, la operabilidad, la disponibilidad y el mantenimiento son generales en casi todos los casos de decisión.

- La *confiabilidad* tiene un significado se la probabilidad de que el elemento o sistema en cuestión no falle durante un periodo de tiempo bajo condiciones preescritas.
- La *operabilidad* se refiere a la facilidad con que un diseño determinado puede ser manejado u operado por seres humanos.
- La *disponibilidad* es la proporción de tiempo que una máquina está en condiciones de ser utilizada y, por lo tanto, en que no está “fuera de servicio”

por reparación, mantenimiento u otras formas de atención. Este criterio es especialmente importante cuando se ha invertido una gran cantidad de dinero en la máquina. Si se invierten 8 millones de dólares en un gran aeroplano comercial, es de desearse que su disponibilidad sea máxima. También es importante cuando la gente depende en alto grado de un sistema, como sucede con los servicios públicos de abastecimiento de agua, los sistemas energía de los hospitales y los ascensores de un edificio de 40 pisos.

- El *mantenimiento* define los costos y tiempos necesarios para que la solución se ajuste a modificaciones o fallos predecibles en piezas o al exponerse condiciones extremas.

Especificación de una solución

Los datos de entrada a esta fase son la solución elegida, parte de ella en forma de bosquejo, apuntes, cálculos, etcétera, y gran parte de ella todavía en la cabeza del ingeniero. Además de ser incompleto, este material está desorganizado y difícilmente en condiciones de poder ser presentado a los jefes o a los clientes.

Falta describir con los detalles suficientes los atributos físicos, lógicos y las características de funcionamiento de la solución propuesta, de manera que las personas que deben aprobarla, los encargados de su construcción y quienes la manejarán y conservarán, puedan desempeñar satisfactoriamente sus funciones.

Los datos de salida de esta fase consisten usualmente de dibujos del proyecto, un informe escrito y, posiblemente, un modelo físico o iconográfico tridimensional. Los primeros de planos o representaciones pueden ser simplemente dibujos de una solución cuidadosamente realizados; detallados y acotados.

El segundo medio, el informe técnico, suele ser un documento bastante formal que describe la propuesta con palabras, diagramas, planos y referencias técnicas. Este informe también describe el funcionamiento de la solución y proporciona una evaluación cabal de ella. Es probable que esa fase del proceso de diseño comprenda detalles considerables de algunos de sus componentes de los cuales se tenga el conocimiento funcional y la certeza de que serán incorporados en la solución.

Recapitulando, el proceso de diseño es una serie de etapas en la evolución de la solución a un problema. El objeto de cada fase es diferente; asimismo lo será el tipo de actividad en la resolución de problemas que predomine en cada uno. Sin embargo, estas etapas no tienen fronteras bien definidas, ni tampoco constituyen la serie ordenada de pasos concretos y bien definidos que esperaría el idealista. Hay cierta confusión a medida que el énfasis pasa de una fase a la siguiente. Ocasionalmente, las soluciones se le ocurrirán a uno mientras la definición del problema sea la actividad predominante; durante la fase de búsqueda puede decidirse reformular el problema. Similarmente, es imposible no efectuar una cierta evaluación en la fase de búsqueda. El azar juega un papel significativo en este proceso; nuevas informaciones y nuevas ideas se descubre inesperadamente, se revelan consecuencias adversas y se encuentran callejones sin salida.

En algunos casos el proceso de diseño estudia sólo las características generales de la solución, tratando temporalmente los subsistemas y componentes como "cajas negras". Lo anterior se realiza a veces en un estudio de factibilidad, donde los detalles se especifican solamente al grado necesario para permitir al ingeniero predecir satisfactoriamente los costos de desarrollar y producir el dispositivo o sistema y pronosticar la aceptación por los consumidores.

El proceso de diseño (de manera general) se esquematiza en la figura 1.1.

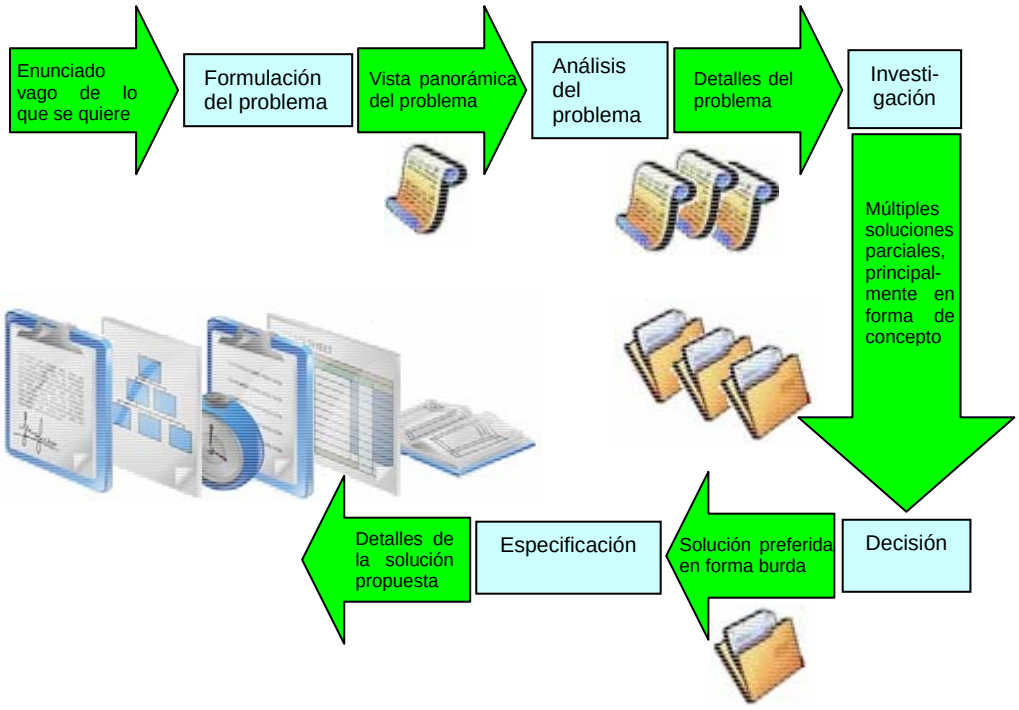


Figura 1.1 Fases del proceso de diseño

## 1.2 Ingeniería de software

Para entender el término de Ingeniería de Software, es necesario por principio abordar la definición del software, para que basado en su descripción la definición de Ingeniería de Software fluya de manera natural.

Cuando se construye hardware, el proceso creativo humano (análisis, diseño, construcción, prueba) se traduce finalmente en una forma física. Si construimos una nueva computadora, nuestro boceto inicial, diagramas formales de diseño y prototipo de prueba, evolucionan hacia un producto físico (circuitos impresos, fuentes de potencia, etc.).

El software es un elemento del sistema que es lógico, en lugar de físico. Por tanto el software tiene características considerablemente distintas a las del hardware.

1.- *El software se desarrolla, no se fabrica en un sentido clásico*. Aunque existen similitudes entre el desarrollo del software y la construcción del hardware, ambas actividades son fundamentalmente diferentes. En ambas actividades la buena calidad se adquiere mediante un buen diseño, pero la fase de construcción del hardware puede introducir problemas de calidad que no existen (o son fácilmente corregibles) en el software.

Los costes del software se encuentran en la ingeniería. Esto significa que los proyectos de software no se pueden gestionar como si fueran proyectos de fabricación.

2.- *El software no se estropea*. La figura 1.2 describe para el hardware la proporción de fallos como una función del tiempo. Esa relación denominada frecuentemente “curva de bañera”, indica que en general el hardware exhibe muchas fallas al principio de su

vida (estos fallos son atribuibles normalmente a defectos del diseño o de la fabricación) esta parte de la gráfica se denomina “Mortalidad infantil”. Una vez corregidos los defectos, la tasa de fallos cae hasta un nivel casi estacionario donde permanece cierto tiempo, los fallos vuelven a presentarse a medida que los componentes del hardware sufren los efectos acumulativos de la suciedad, la vibración, los malos tratos, las temperaturas extremas y muchos otros males externos.

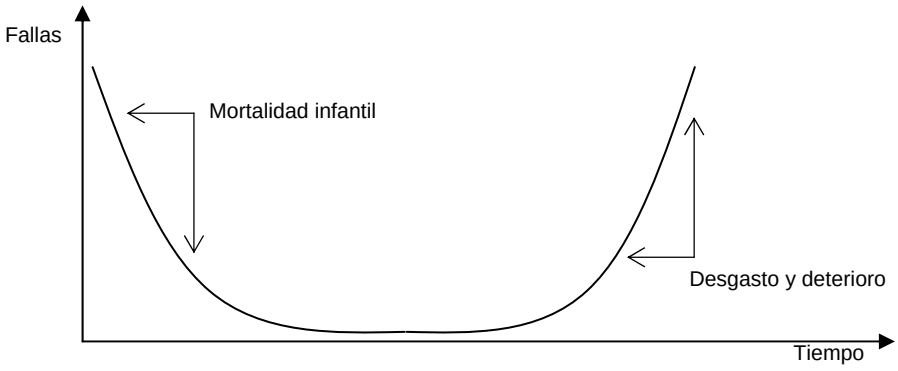


Figura 1.2 Fallas del hardware contra el tiempo

El software no es susceptible a los males del entorno que hacen que el hardware se estropee. Los defectos no detectados en el software harán que este falle durante las primeras etapas de su vida. Sin embargo, una vez que se corrigen y suponiendo que no se introducen nuevos errores, la curva se aplana. La figura 1.3 es una gran simplificación de los modelos reales de fallos del software. Sin embargo la implicación es clara, el software no se estropea, más sin embargo es altamente susceptible a la obsolescencia, además de que es altamente probable que conforme se realizan cambios aparezcan nuevos fallos provocando que la curva presente nuevos picos que descienden cuando se corrigen estos y reaparecen con nuevos requerimientos.

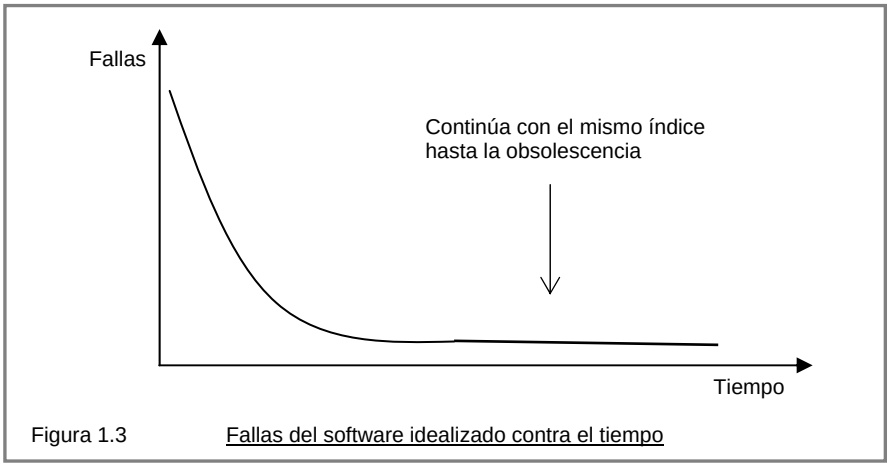


Figura 1.3 Fallas del software idealizado contra el tiempo

Otro aspecto de este deterioro que ilustra la diferencia entre el hardware y el software, es que, cuando un componente de hardware se estropea, se sustituye por una pieza de repuesto. En el software esta pieza de reemplazo no existe. Cada fallo en el software indica un error en el diseño o en el proceso mediante el que se tradujo el diseño a código máquina ejecutable. Por tanto el mantenimiento del software tiene una complejidad considerablemente mayor que la del hardware.

3.- *La mayoría del software se construye a medida, en vez de ensamblar componentes existentes*. Consideremos la forma en la que se diseña y se construye el hardware de control para un producto basado en microprocesador. El ingeniero de diseño construye un sencillo esquema de la circuitería digital, hace algún análisis fundamental para asegurar que se realiza la función adecuada y va al catálogo de ventas de componentes digitales existentes. Cada circuito integrado tiene un número de pieza, una función definida y válida, una interfaz bien definida y un conjunto estándar de criterios de integración. Después de seleccionar cada componente, puede solicitarse la compra.

Por desgracia en el caso del software esto no siempre esta disponible y la utilización de clases y bibliotecas son una solución pero traen consigo carga de código que no se utiliza, es decir, puede darse en caso de que de una biblioteca extensa solo ocupemos una función o clase y todo lo demás de ella no se ocupe.

La reutilización es una característica importante para un componente de software de alta calidad. En los años sesenta se construyeron bibliotecas de subrutinas reutilizables en una amplia serie de aplicaciones científicas y de ingeniería. Esas bibliotecas de subrutinas reutilizaban de forma efectiva algoritmos bien definidos, pero tenían un dominio de aplicación limitado. Hoy en día, hemos extendido nuestra visión de reutilización para abarcar no sólo los algoritmos, sino también estructuras de datos. Los componentes reutilizables modernos encapsulan tanto datos como procesos que se aplican a los datos, permitiendo al ingeniero de software crear nuevas aplicaciones a partir de las partes reutilizables.

Las estructuras de datos y los detalles de procesamiento requeridos para construir la interfaz están contenidos en una biblioteca de componentes reutilizables orientados a la construcción de interfaces.

Los componentes de software se construyen mediante un lenguaje de programación que tiene un vocabulario limitado, una sintaxis y semántica establecidas.

Los lenguajes máquina son una representación simbólica del conjunto de instrucciones de la CPU. Si un buen programador produce programas de fácil mantenimiento y bien documentados, puede utilizar el lenguaje máquina para hacer uso extremadamente eficiente de la memoria y para optimizar la velocidad de ejecución del programa. Si el programa esta mal diseñado y tiene poca documentación, le lenguaje máquina puede convertirse en una pesadilla.

Los lenguajes de alto nivel permiten al programador y al programa independizarse de la máquina. Cuando se utiliza un traductor sofisticado, el vocabulario, la gramática, la sintaxis y la semántica de un lenguaje de alto nivel pueden ser mucho más sofisticados que los lenguajes máquina. De hecho, los compiladores e intérpretes de los lenguajes de alto nivel producen lenguaje máquina como salida.

Hoy en día el software tiene un doble papel, es un producto y al mismo tiempo, el vehículo para hacer entrega de un producto. Como producto, hace entrega de la potencia informática del hardware informático, ya sea que resida dentro de un teléfono celular u opere dentro de una computadora, el software es un transformador de información, produciendo, gestionando, adquiriendo, modificando, mostrando o transmitiendo información. Dicha información podría ser tan simple como la unidad más simple y básica como por ejemplo un bit o tan compleja y extensa como una presentación multimedia con imágenes de alta resolución. Como vehículo utilizado para hacer entrega del producto, el software actúa como la base de control de la

computadora (sistemas operativos), la comunicación de información (redes), y la creación y control de otros programas (herramientas de software y entornos).

El software hace entrega de lo que hace algún tiempo se analizó y predijo sería el producto más importante del siglo veintiuno y en la actualidad reconocemos como un hecho fáctico, la información. El software transforma datos personales (p. ej.: transacciones financieras de una persona) para que los datos sean más útiles en un contexto local; gestiona información comercial para mejorar la competitividad; proporciona el acceso a redes de información por todo el mundo (p. ej.: Internet y los protocolos tan numerosos y diversos que sobre el operan); y también proporciona el medio para adquirir información en todas sus formas.

El papel del software informático ha sufrido un cambio significativo en la segunda mitad del siglo veinte, enormes mejoras en el rendimiento del hardware, profundos cambios en la arquitectura informática, grandes aumentos en la capacidad de almacenamiento (llámese este memoria de procesamiento principal secundaria, de registro o de almacenamiento permanente o semipermanente), gran variedad de opciones de entrada y salida, e incluso las necesidades derivadas de la extensión en el número de usuarios, tendencias comerciales, tendencias políticas o tendencias de seguridad, han precipitado la creación de sistemas más sofisticados y complejos basados en computadoras. La sofisticación y la complejidad pueden producir resultados deslumbrantes cuando un sistema tiene éxito, pero también pueden suponer grandes problemas y riesgos para aquellos que deben construir dichos sistemas.

Durante los primeros años de la era computadora, el software se contemplaba como un añadido. La programación de computadoras era considerada un arte para el que existían pocos métodos sistemáticos. El desarrollo del software se realizaba virtualmente sin ninguna planificación, hasta que los planes comenzaron a descalabrarse y los costes a aumentar sobrepasando los topes. Los programadores trataban de hacer las cosas bien, y con un esfuerzo heroico, a menudo salían con éxito.

Durante los primeros años lo normal era que el hardware fuera de propósito general. Por otra parte, el software se diseñaba a medida para cada aplicación y tenía una distribución relativamente pequeña. El software como producto (es decir, programas para ser vendidos a uno o más clientes), estaba en su infancia. La mayoría del software que se desarrollaba era utilizado solo por unos pocos individuos o empresas. En el procesos de desarrollo del software a menudo la misma persona lo escribía, lo ejecutaba y, si fallaba, lo depuraba ella misma (esto en el caso ideal de que se depurara), o en algunas ocasiones los vicios o errores permanecían ocultos hasta que los errores surgían por si solos, sin embargo la movilidad de trabajo era relativamente baja, por lo que los ejecutivos confiaban en que si algún error surgía, la persona o desarrollador estaría ahí para resolver el problema. Es así que el entorno de desarrollo era personalizado, el diseño era un proceso implícito, realizado en la mente de alguien y por supuesto la documentación referente a dichos diseños y desarrollos no existía.

A lo largo de los primeros años se aprendió mucho sobre la implementación de sistemas informáticos, pero relativamente poco sobre la ingeniería de las computadoras. Sin embargo y en honor a la verdad, se debe reconocer que durante esa era se desarrollaron muchos sistemas informáticos excepcionales. Algunos de los cuales se siguen utilizando y son admirados hasta el día de hoy.
La segunda era en evolución de los sistemas de computadoras se extiende desde la mitad de la década de los sesenta hasta finales de los setentas. La multiprogramación y los sistemas multiusuario introdujeron nuevos conceptos de interacción hombre-

máquina. Las técnicas interactivas abrieron un muevo mundo de aplicaciones y nuevos niveles de sofisticación del hardware y del software. Los sistemas de tiempo real podían recoger, analizar y transformar datos de múltiples fuentes, controlando así los procesos y produciendo salidas en milisegundos en lugar de minutos. Los avances en dispositivos de almacenamiento en línea condujeron a la primera generación de sistemas de gestión de bases de datos.

La segunda era se caracterizó también por el establecimiento del software como producto. El software se desarrollaba ya para tener una amplia distribución en un mercado multidisciplinar. Los programas se distribuían para computadoras grandes o para mini computadoras, a cientos e incluso a miles de usuarios.

Conforme se crecía el número de sistemas informáticos, comenzaron a extenderse las bibliotecas de software de computadora.

Más sin embargo un problema empezó a vislumbrarse en el horizonte. Los programas contenían decenas de miles de sentencias fuentes. Todos esos programas, todas esas sentencias fuente tenían que ser corregidos cuando se detectaban fallos, modificadas cuando cambiaban los requisitos de los usuarios o adaptarlos a nuevos dispositivos de hardware que se hubiera adquirido. Estas actividades se denominaron colectivamente, mantenimiento de software. Desgraciadamente la complejidad y alta relación entre las partes que componían el todo de los programas, comenzó a absorber recursos en una medida alarmante.

Aún peor, la naturaleza personalizada de muchos programas los hacia virtualmente imposible de mantener. Había comenzado una crisis del software.

La tercera era en la evolución de los sistemas de cómputo comenzó a mediados de los años setenta y continuó más allá de una década. El sistema distribuido, múltiples computadoras cada una ejecutando funciones concurrentemente y comunicándose con alguna otra, incremento notablemente la complejidad de los sistemas informáticos. Las redes de área local, de área amplia y de área global, las comunicaciones digitales de alto ancho de banda y la creciente demanda de acceso instantáneo a los datos, supusieron una fuerte presión sobre los desarrolladores del software. Aún más los sistemas y el software que lo permitían continuaron residiendo dentro de la industria y de la academia. El uso personal era realmente extraño.

La conclusión de la tercera era se caracterizó por la llegada y amplio uso de los microprocesadores. El microprocesador produjo un extenso grupo de productos inteligentes. Desde automóviles hasta hornos de microondas, desde robots industriales a equipos de diagnóstico de suero sanguíneo, pero ninguno más importante que la computadora personal. En menos de una década, pasaron de ser dispositivos exclusivos de la industria, la educación e investigación y la milicia, ha ser un producto de fácil acceso al público.

La cuarta era de la evolución de sistemas informáticos se aleja de las computadoras individuales y de los programas de computadora, dirigiéndose al impacto colectivo de las computadoras y del software. Potentes máquinas personales controladas por sistemas operativos sofisticados, en redes globales y locales, acompañadas por aplicaciones de software avanzadas se han convertido en la norma. Las arquitecturas informáticas están cambiando de entornos centralizados de grandes computadoras a entornos descentralizados cliente/servidor. Las redes de información en todo el mundo proporcionan la súper infraestructura que hace de Internet la superautopista de la información, haciendo que Internet se pueda considerar como un software al que pueden acceder usuarios individuales. A medida que la cuarta generación progresó

surgieron nuevas tecnologías. Las tecnologías orientadas a objetos tomaron rápidamente su importante pedazo del pastel dentro del mundo de los sistemas.

La quinta generación de computadoras fue un proyecto ambicioso lanzado por Japón a finales de los 70. Su objetivo era el desarrollo de una clase de computadoras que utilizarían técnicas de inteligencia artificial al nivel del lenguaje de máquina y serían capaces de resolver problemas complejos, como la traducción automática de una lengua natural a otra (del japonés al inglés, por ejemplo).

El proyecto duró diez años, pero no obtuvo los resultados esperados: las computadoras actuales siguieron así, ya que hay muchos casos en los que, o bien es imposible llevar a cabo una paralelización de las tareas, procesos o cálculos del mismo, o una vez llevado a cabo ésta, no se aprecia mejora alguna, o en el peor de los casos, se produce una pérdida de rendimiento. Hay que tener claro que para realizar un programa paralelo debemos, para empezar, identificar dentro del mismo partes que puedan ser ejecutadas por separado en distintos procesadores. Además, es importante señalar que un programa que se ejecuta de manera secuencial, debe recibir numerosas modificaciones para que pueda ser ejecutado de manera paralela, es decir, primero sería interesante estudiar si realmente el trabajo que esto nos llevará se ve compensado con la mejora del rendimiento de la tarea después de paralelizarla.

A través de las múltiples generaciones desde los años 50, Japón había sido el seguidor en términos del adelanto y construcción de las computadoras de los modelos de los Estados Unidos y el Reino Unido. Japón decidió romper con esta naturaleza de seguir a los líderes y a mediados de la década de los 70 comenzó a abrirse camino hacia un futuro en la industria de informática. El centro del desarrollo y proceso de la información de Japón fue el encargado llevar a cabo un plan para desarrollar el proyecto. En 1979 ofrecieron un contrato de tres años para realizar estudios más profundos junto con industria y la academia. Fue durante este período cuando el término "computadora de quinta generación" comenzó a ser utilizado.

Los campos principales para la investigación de este proyecto inicialmente eran:

- Tecnologías para el proceso del conocimiento.
- Tecnologías para procesar bases de datos y bases de conocimiento masivo.
- Sitios de trabajo del alto rendimiento.
- Informáticas funcionales distribuidas.
- Supercomputadoras para el cálculo científico.

Debido a la conmoción suscitada que causó que los japoneses fueran exitosos en el área de los artículos electrónicos durante la década de los 70, y que prácticamente hicieran lo mismo en el área de la automoción durante los 80, el proyecto de la quinta generación tuvo mucha reputación entre los otros países.

Tal fue su impacto que se crearon proyectos paralelos. En Estados Unidos, la Corporación de Microelectrónica y Tecnologías de la Computación, en Inglaterra fue Alves, y en Europa su reacción fue conocida como el Programa Europeo en Investigación Estratégica de la Tecnología de la Información.

Como uno de los productos finales del Proyecto se desarrollaron 5 máquinas de Inferencia Paralela (PIM) teniendo como una de sus características principales 256 elementos de Procesamiento Acoplados en red. El proyecto también produjo herramientas que se podían utilizar con estos sistemas tales como el Sistema Paralelo

de Gerencia de Bases de Datos Kappa, el Sistema de Razonamiento Legal HELIC-II y el Teorema Autómata de Aprobaciones MGTP.

Aún y cuando los resultados de la quinta generación no fueron los esperados, grandes avances han resultado de está. La paralelización es en ciertas ramas del cómputo una solución de gran importancia, por ejemplo en el procesamiento orientado a la visualización de programas altamente complejos con millones variables los cuales resultarían en tiempo práctico imposibles de resolver como el de la simulación del clima mundial. La filosofía y técnicas de los sistemas expertos y el software de inteligencia artificial son también un legado importante de esta generación.

La ingeniería de software se puede considerar como la ingeniería aplicada al software, esto es en base a herramientas preestablecidas, la aplicación de las mismas de la forma más eficiente y óptima; objetivos que siempre busca la ingeniería. No es sólo de la resolución de problemas, sino más bien teniendo en cuenta las diferentes soluciones, elegir la más apropiada.

De manera un poco más formal, el término Ingeniería de software definido por la IEEE es la suma total de los programas de computadora, procedimientos, reglas, la documentación asociada y los datos que pertenecen a un sistema de cómputo". La Ingeniería de Software (SE del inglés *Software Engineering*) es un enfoque sistemático del desarrollo, operación, mantenimiento y retiro del software, que en palabras más llanas, se considera que la “Ingeniería de Software” es la rama de la ingeniería que aplica los principios de la ciencia de la computación y las matemáticas para lograr soluciones costo-efectivas (eficaces en costo o económicas) a los problemas de desarrollo de software", es decir, "permite elaborar consistentemente productos correctos, utilizables, costo-efectivos y que funcione sobre máquinas reales.

El término “ingeniería de software” fue introducido a finales de los 60 a raíz de la crisis del software. Esta crisis fue el resultado de la introducción de la tercera generación del hardware, en el que el hardware dejo de ser un impedimento para el desarrollo de la informática; reduciendo los costos y mejorando la calidad y eficiencia en el software producido.

La crisis del software se caracterizó por los siguientes problemas:

- Imprecisión en la planificación del proyecto y estimación de los costos.
- Baja calidad del software.
- Productos de software no confiables.
- Dificultad de mantenimiento de programas con un diseño poco estructurado, etc.

El proceso de ingeniería de software se define como un conjunto de etapas parcialmente ordenadas con la intención de lograr un objetivo, en este caso, la obtención de un producto de software de calidad. El proceso de desarrollo de software es aquel en que las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos transformados en diseño y el diseño implementado en código, el código es probado, documentado y certificado para su uso operativo. Concretamente define quién está haciendo qué, cuándo hacerlo y cómo alcanzar un cierto objetivo.

Objetivos de la ingeniería de software

En la construcción y desarrollo de proyectos se aplican métodos y técnicas para resolver los problemas, la informática aporta herramientas y procedimientos sobre los que se apoya la ingeniería de software.

- Mejorar la calidad de los productos de software.
- Aumentar la productividad y trabajo de los ingenieros del software.
- Facilitar el control del proceso de desarrollo de software.
- Suministrar a los desarrolladores las bases para construir software de alta calidad en una forma eficiente.
- Definir una disciplina que garantice la producción y el mantenimiento de los productos software desarrollados en el plazo fijado y dentro del costo estimado.

Los sistemas computacionales desarrollados a partir de una Ingeniería de software colaboran en la mejora de cinco elementos clave para las empresas u organizaciones, a menudo conocidos como las cinco C's.

Capacidad

Las actividades de la organización están influenciadas por la capacidad de ésta para procesar transacciones con rapidez y eficiencia. Los sistemas de información mejoran esta capacidad en tres formas.

Aumentan la velocidad de procesamiento:

- Los sistemas basados en computadora pueden ser de ayuda para eliminar la necesidad de cálculos tediosos y comparaciones repetitivas.
- Un sistema automatizado puede ser de gran utilidad si lo que se necesita es un procesamiento acelerado.

Aumento en el volumen:

- La incapacidad para mantener el ritmo de procesamiento no significa el abandono de los procedimientos existentes. Quizá éstos resulten inadecuados para satisfacer las demandas actuales. En estas situaciones el analista de sistemas considera el impacto que tiene la introducción de procesamiento computarizado, si el sistema existente es manual. Es poco probable que únicamente el aumento de la velocidad sea la respuesta. El tiempo de procesamiento por transacción aumenta si se considera la cantidad de actividades comerciales de la empresa junto con su patrón de crecimiento.

Recuperación más rápida de la información:

- Las organizaciones almacenan grandes cantidades de datos, por eso, debe tenerse en cuenta donde almacenarlos y como recuperarlos cuando se los necesita. Cuando un sistema se desarrolla en forma apropiada, se puede recuperar en forma rápida la información.

Costo

Vigilancia de los costos:

- Para determinar si la compañía evoluciona en la forma esperada, de acuerdo con lo presupuestado, se debe llevar a cabo el seguimiento de los costos de mano de obra, bienes y gastos generales.
- La creciente competitividad del mercado crea la necesidad de mejores métodos para seguir los costos y relacionarlos con la productividad individual y organizacional.

Reducción de costos:

- Los diseños de sistemas ayudan a disminuir los costos, ya que toman ventaja de las capacidades de cálculo automático y de recuperación de datos que están incluidos en procedimientos de programas en computadora. Muchas tareas son realizadas por programas de cómputo, lo cual deja un número muy reducido de éstas para su ejecución manual, disminuyendo al personal.

Control

Mayor seguridad de información:

- Algunas veces el hecho de que los datos puedan ser guardados en una forma adecuada para su lectura por medio de una máquina, es una seguridad difícil de alcanzar en un medio ambiente donde no existen computadoras.
- Para aumentar la seguridad, generalmente se desarrollan sistemas de información automatizados. El acceso a la información puede estar controlado por un complejo sistemas de contraseñas, limitado a ciertas áreas o personal, si está bien protegido, es difícil de acceder.

Menor margen de error: (mejora de la exactitud y la consistencia)

- Esto se puede lograr por medio del uso de procedimientos de control por lotes, tratando de que siempre se siga el mismo procedimiento. Cada paso se lleva a cabo de la misma manera, consistencia y con exactitud: por otra parte se efectúan todos los pasos para cada lote de transacciones. A diferencia del ser humano, el sistema no se distrae con llamadas telefónicas, ni olvidos e interrupciones que sufre el ser humano. Si no se omiten etapas, es probable que no se produzcan errores.

Comunicación

La falta de comunicación es una fuente común de dificultades que afectan tanto a cliente como a empleados. Sin embargo, los sistemas de información bien desarrollados amplían la comunicación y facilitan la integración de funciones individuales.

Interconexión: (aumento en la comunicación)

- Muchas empresas aumentan sus vías de comunicación por medio del desarrollo de redes para este fin, dichas vías abarcan todo el país y les

permiten acelerar el flujo de información dentro de sus oficinas y otras instalaciones que no se encuentran en la misma localidad.

- Una de las características más importantes de los sistemas de información para oficinas es la transmisión electrónica de información, como por ejemplo, los mensajes y los documentos.

Integración de áreas en las empresas:

Con frecuencia las actividades de las empresas abarcan varias áreas de la organización, la información que surge en un área se necesita en otra área, por ejemplo.

- Los sistemas de información ayudan a comunicar los detalles del diseño a los diferentes grupos, mantienen las especificaciones esenciales en un sitio de fácil acceso y calculan factores tales como el estrés y el nivel de costos a partir de detalles proporcionados por otros grupos.

Calidad

El concepto de calidad en los productos de software debe formularse de forma particular. Primero es conveniente indicar sus características diferenciadoras frente a otros productos: el software se desarrolla, no se fabrica en el sentido clásico; es inmaterial y no se deteriora con el uso o el tiempo (aunque tiene un ciclo de vida); su fiabilidad es difícil de comprobar; la mayoría del software se construye a medida y necesita de actualización permanente y en la mayoría de los casos es dependiente del entorno donde se ejecuta

La calidad, puede definirse como un conjunto de características o cualidades, tales como: eficiencia, fiabilidad, utilidad, funcionalidad, facilidad de mantenimiento, portabilidad, etc., variando la importancia de cada una de ellas de un producto a otro.

En las empresas de Ingeniería de software, la calidad se obtiene mejorando día a día el proceso de producción, mantenimiento y gestión del software. Para optimizar la calidad de los productos y/o servicios es preciso conocer al cliente y sus necesidades, conocer a la competencia y poseer un modelo de calidad. Esto último permitirá incrementar la fiabilidad, reducir el mantenimiento, aumentar la satisfacción del cliente, mejorar la dirección del proyecto, detectar errores pronto / rápido e incrementar el beneficio. Pero también la madurez de los procesos de desarrollo y las técnicas adecuadas para mejorarlos y tener una gestión adecuada de las mejoras de dichos procesos.

La ingeniería de software requiere llevar a cabo muchas tareas, sobre todo las siguientes:

Análisis de requisitos

Extraer los requisitos de un producto de software es la primera etapa para crearlo. Mientras que los clientes piensan que ellos saben lo que el software tiene que hacer, se requiere de habilidad y experiencia en la ingeniería de software para reconocer requisitos incompletos, ambiguos o contradictorios. El resultado del análisis de requisitos con el cliente se plasma en el documento ERS, Especificación de Requerimientos del Sistema, cuya estructura puede venir definida por varios estándares, tales como CMM-I. Asimismo, se define un diagrama de Entidad/Relación,

en el que se plasman las principales entidades que participarán en el desarrollo del software.

Especificación

Es la tarea de describir detalladamente el software a ser escrito, en una forma matemáticamente rigurosa. En la realidad, la mayoría de las buenas especificaciones han sido escritas para entender y afinar aplicaciones que ya estaban desarrolladas. Las especificaciones son más importantes para las interfaces externas, que deben permanecer estables.

Diseño y arquitectura

Se refiere a determinar como funcionará de forma general sin entrar en detalles. Según Yourdon consiste en incorporar consideraciones de la implementación tecnológica, como el hardware, la red, etc. Se definen los casos de uso para cubrir las funciones que realizará el sistema, y se transforman las entidades definidas en el análisis de requisitos en clases de diseño, obteniendo un modelo cercano a la programación orientada a objetos.

Programación

Reducir un diseño a código puede ser la parte más obvia del trabajo de ingeniería de software, pero no es necesariamente la porción más larga.

Prueba

Consiste en comprobar que el software realice correctamente las tareas indicadas en la especificación. Una técnica de prueba es probar por separado cada módulo del software, y luego probarlo de forma integral.

Documentación

Realización del manual de usuario, y posiblemente un manual técnico con el propósito de mantenimiento futuro y ampliaciones al sistema.

Mantenimiento

Mantener y mejorar el software para enfrentar errores descubiertos y nuevos requisitos. Esto puede llevar más tiempo incluso que el desarrollo inicial del software. Alrededor de 2/3 de toda la ingeniería de software tiene que ver con dar mantenimiento. Una pequeña parte de este trabajo consiste en arreglar errores, o bugs. La mayor parte consiste en extender el sistema para hacer nuevas cosas. De manera similar, alrededor de 2/3 de toda la ingeniería civil, arquitectura y trabajo de construcción es dar mantenimiento.

Finalmente la Ingeniería de Software de modo práctico tiene que ver con muchos campos en diferentes formas:

Matemáticas

Los programas tienen muchas propiedades matemáticas. Por ejemplo la corrección y la complejidad de muchos algoritmos son conceptos matemáticos que pueden ser rigurosamente probados. El uso de matemáticas en la Ingeniería de Software es llamado *métodos formales*. Edsger Dijktravistas pragmáticos y las características

esperadas de los ingenieros. Análisis, documentación, y código comentado son signos de un ingeniero. Bazzano y Ostrichi han argumentado que es una ingeniería.

Manufactura

Los programas son construidos en una secuencia de pasos. El hecho de definir propiamente y llevar a cabo estos pasos, como en una línea de ensamblaje, es necesario para mejorar la productividad de los desarrolladores y la calidad final de los programas. Este punto de vista inspira los diferentes procesos y metodologías que encontramos en la IS.

Manejo de Proyectos

El software comercial (y mucho no comercial) requiere manejo de proyectos. Hay presupuestos y calendarizaciones establecidas. Gente para liderar. Recursos (espacio de oficina, computadoras) por adquirir. Todo esto encaja apropiadamente con la visión del Manejo de Proyectos.

Arte

Los programas contienen muchos elementos artísticos. Las interfaces de usuario, la codificación, etc. Incluso la decisión para un nombre de una variable o una clase. Donald Knuth es famoso porque ha argumentado que la programación es un arte

Desarrollo de software

La ingeniería de software tiene varios modelos o paradigmas de desarrollo en los cuales se puede apoyar para la realización de software, de los cuales podemos destacar a éstos por ser los más utilizados y los más completos:

- Modelo en cascada (ciclo de vida clásico).
- Modelo en espiral.
- Modelo de prototipos.
- Método en V.
- Desarrollo por etapas.

### 1.2.1 Paradigmas de programación

Para resolver los problemas reales en una industria, un ingeniero de software o un equipo de ingenieros deben incorporar una estrategia de desarrollo que acompañe al proceso, métodos y capas de herramientas, a esta estrategia a menudo se le llama modelo de procesos o paradigma de ingeniería del software

Un paradigma de programación representa un enfoque particular o filosofía para la construcción del software. Cada uno tiene ventajas y desventajas. También hay situaciones donde un paradigma en particular resulta más apropiado que otro para la resolución de un problema.

Un paradigma de programación es un modelo básico de diseño y desarrollo de programas, que permite producir programas con unas directrices específicas, tales como: estructura modular, fuerte cohesión, alta rentabilidad, etc.

Para algunos puede resultar sorprendente que existan varios paradigmas de programación. La mayor parte de los programadores están familiarizados con un único paradigma. Sin embargo hay multitud de ellos atendiendo a alguna particularidad metodológica o funcional, como por ejemplo el basado en reglas de gran aplicación en la ingeniería del conocimiento para el desarrollo de sistemas expertos, en que el núcleo del mismo son las reglas de producción del tipo "if then"; el de programación lógica, basado en asertos y reglas lógicas que define un entorno de programación de tipo conversacional, deductivo, simbólico y no determinista; el de programación funcional, basado en funciones, forma funcionales para crear funciones y mecanismos para aplicar los argumentos, y que define un entorno de programación interpretativo, funcional y aplicativo, el de programación heurística que aplica para la resolución de los problemas "reglas de buena lógica" que presentan visos de ser correctas aunque no se garantiza su éxito, modelando el problema de una forma adecuada para aplicar estas heurísticas atendiendo a su representación, estrategias de búsqueda y métodos de resolución; el de programación paralela; el basado en restricciones; el basado en el flujo de datos; el orientado al objeto, etc.

Un paradigma de programación es una colección de modelos conceptuales que juntos modelan el proceso de diseño y determinan, al final, la estructura de un programa. Esa estructura conceptual de modelos está pensada de forma que esos modelos determinan la forma correcta de los programas y controlan el modo en que pensamos y formulamos soluciones, y al llegar a la solución, ésta se debe de expresar mediante un lenguaje de programación. Para que este proceso sea efectivo las características del lenguaje deben reflejar adecuadamente los modelos conceptuales de ese paradigma.

Cuando un lenguaje refleja bien un paradigma particular, se dice que soporta el paradigma, y en la práctica un lenguaje que soporta correctamente un paradigma, es difícil distinguirlo del propio paradigma, por lo que se identifica con él.

Tipos de paradigmas

Los paradigmas de programación se ubican en tres categorías:

a) Los que soportan técnicas de programación de bajo nivel (ej.: copia de archivos frente estructuras de datos compartidos).
b) Los que soportan métodos de diseño de algoritmos (ej.: divide y vencerás, programación dinámica, etc.).
c) Los que soportan soluciones de programación de alto nivel, como los descritos en el punto anterior.

Los paradigmas de programación de alto nivel se agrupan en tres categorías de acuerdo con la solución que aportan para resolver el problema:

a) Solución procedimental u operacional. Describe etapa a etapa el modo de construir la solución. Es decir señala la forma de obtener la solución.
b) Solución demostrativa. Es una variante de la procedimental. Especifica la solución describiendo ejemplos y permitiendo que el sistema generalice la solución de estos ejemplos para otros casos. Aunque es fundamentalmente procedimental, el hecho de producir resultados muy diferentes a ésta, hace que sea tratada como una categoría separada.
c) Solución declarativa. Señala las características que debe tener la solución, sin describir cómo procesarla. Es decir señala qué se desea obtener pero no cómo obtenerlo.

Paradigmas procedimentales u operacionales

La característica fundamental de estos paradigmas es la secuencia computacional realizada etapa a etapa para resolver el problema. Su mayor dificultad reside en determinar si el valor computado es una solución correcta del problema, por lo que se han desarrollado multitud de técnicas de depuración y verificación para probar la corrección de los problemas desarrollados basándose en este tipo de paradigmas.

Pueden ser de dos tipos básicos: Los que actúan modificando repetidamente la representación de sus datos (efecto de lado); y los que actúan creando nuevos datos continuamente (sin efecto de lado).

Los paradigmas con efecto de lado utilizan un modelo en el que las variables están estrechamente relacionadas con direcciones de la memoria. Cuando se ejecuta el programa, el contenido de estas direcciones se actualiza repetidamente, pues las variables reciben múltiples asignaciones, y al finalizar el trabajo, los valores finales de las variables representan el resultado.

Existen dos tipos de paradigmas con efectos de lado:

- El imperativo.
- El orientado a objetos.

Los paradigmas procedimentales definen la secuencia explícitamente, pero esta secuencia se puede procesar en serie o en paralelo. En este segundo caso el procesamiento paralelo puede ser asíncrono (procesos simples aplicados simultáneamente a muchos objetos) o síncrono (cooperación de procesos paralelos).

Paradigmas declarativos

En este tipo, un programa se construye señalando hechos, reglas, restricciones, ecuaciones, transformaciones y otras propiedades derivadas del conjunto de valores que configuran la solución.

A partir de esta información el sistema debe de proporcionar un esquema que incluya el orden de evaluación que compute una solución. Aquí no existe la descripción de las diferentes etapas a seguir para alcanzar una solución, como en el caso anterior.

Estos paradigmas permiten el uso de variables para almacenar valores intermedios, pero no para actualizar estados de información.

Dado que estos paradigmas especifican la solución sin indicar cómo construirla, en principio eliminan la necesidad de probar que el valor calculado es el valor solución.

En principio, los paradigmas declarativos no son soluciones inherentes de tipo serie o paralelo, ya que no dirigen la secuencia de control y no pueden alterar el natural no paralelismo del algoritmo. No obstante, los paradigmas pseudodeclarativos requieren al menos un limitado grado de secuencia, y por lo tanto admiten versiones en serie y paralelo.

Paradigmas demostrativos

Cuando se programa bajo un paradigma demostrativo (también llamada programación por ejemplos), el programador no especifica procedimentalmente cómo construir una

solución. En su lugar, presentan soluciones de problemas similares y permite al sistema que generalice una solución procedimental a partir de estas demostraciones.

Los esquemas individuales para generalizar tales soluciones van desde simular una secuencia procedimental o inferir intenciones.

Los sistemas que infieren, intentan generalizar usando razonamiento basado en el conocimiento. Una solución basada en la inferencia intenta determinar en qué son similares un grupo de datos u objetos, y, a partir de ello, generalizar estas similitudes.

Otra solución es la programación asistida: el sistema observa acciones que el programador ejecuta, y si son similares o acciones pasadas, intentará inferir cuál es la próxima acción que hará el programador.

Las dos principales objeciones al sistema de inferencia son:

- Si no se comprueban exhaustivamente pueden producir programas erróneos que trabajan correctamente con los ejemplos de prueba, pero que fallen posteriormente en otros casos
- La capacidad de inferencia es tan limitada, que el usuario debe de guiar el proceso en la mayoría de los casos.

Los resultados más satisfactorios de los sistemas de inferencia son en áreas limitadas, donde el sistema tenía un conocimiento semántico importante de la aplicación.

El mayor problema que se presenta con estos sistemas, es conocer cuándo un programa es correcto. En el caso de los sistemas procedimentales, se consigue estudiando el algoritmo y el resultado de juegos de ensayo apropiados.

En el caso de los sistemas demostrativos el algoritmo se mantiene en una representación interna, y su estudio se sale del ámbito de estos sistemas. Por lo que la veracidad de la decisión se debe hacer exclusivamente sobre la base de la eficiencia del algoritmo sobre los casos específicos de prueba.

La programación demostrativa es del tipo "bottom-up" y se adapta bien a nuestra capacidad de pensar. Sin embargo en la mayor parte de los paradigmas la resolución del problema se efectúa aplicando métodos abstractos "top-down".

Todo el desarrollo del software se puede caracterizar como un ciclo de resolución de problemas (figura 1.4) en el que se encuentran cuatro etapas distintas: estatus quo, definición de problemas, desarrollo técnico e integración de soluciones. Status Quo, representa el estado actual de sucesos, la definición de problema, identifica el problema específico a resolver, el desarrollo técnico resuelve el problema a través de la aplicación de alguna tecnología, y finalmente la integración de soluciones ofrece los resultados (p. ej.: documentos, programas, datos, nueva función comercial, producto nuevo) a los que solicitan la solución en primer lugar.

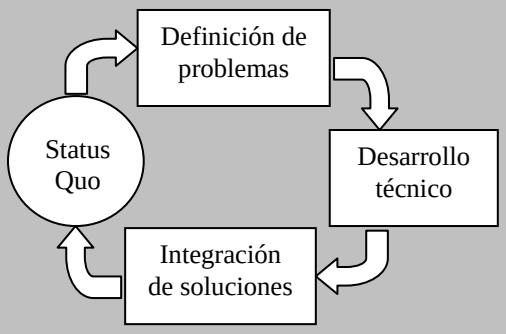


Figura 1.4 Fases de un ciclo de resolución de problemas

El ciclo de resolución de problemas descrito anteriormente, se aplica al trabajo de ingeniería del software en muchos niveles diferentes de resolución. Se puede utilizar en el macro nivel cuando se tiene en consideración la aplicación entera; en un nivel medio cuando se esta considerando los componentes del programa, e incluso en la línea del nivel de código. Por consiguiente se puede utilizar una representación fractal para proporcionar una visión idealizada del proceso. En la figura 1.5 cada etapa de ciclo de resolución de problemas contiene un ciclo idéntico de solución de problemas, el cual contiene todavía otro ciclo (éste llega hasta algún límite racional; para el software, una línea de código).

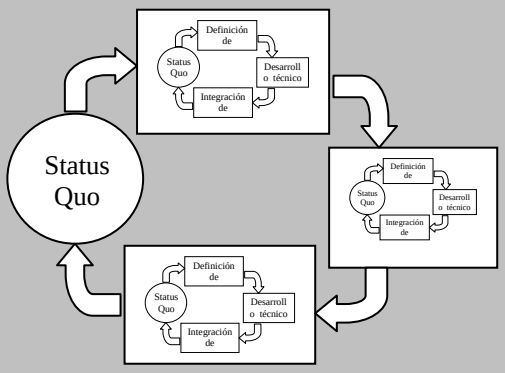


Figura 1.5 Fases dentro de las fases un ciclo de resolución de problemas

En realidad, es difícil particionar actividades de maneta tan nítida como la figura 1.5 da a entender, debido a que existen las etapas se traslapan en algunos partes, aunque esta visión simplificada lleva a una idea muy importante: con independencia del modelo de procesos que se seleccione para un proyecto de software, todas la etapas-status quo, definición de problemas, desarrollo técnico e integración de soluciones – coexisten simultáneamente en un grado mayor o menor de detalle y dada la naturaleza recursiva de la Figura 1.5, las cuatro etapas se aplican igualmente al análisis de una aplicación completa que a la generación de un pequeño segmento de código.

### 1.2.2 Modelo lineal secuencial o cascada

La figura 1.6 ilustra el modelo lineal secuencial para la ingeniería del software. Llamado algunas veces ciclo de vida básico ó modelo en cascada, el modelo lineal secuencial sugiere un enfoque sistemático, secuencial del desarrollo del software que comienza en un nivel de sistemas y progresa con el análisis, diseño, codificación, pruebas y mantenimiento. Modelado según el ciclo de ingeniería convencional, el modelo lineal secuencial se acompaña de las siguientes actividades siguientes:

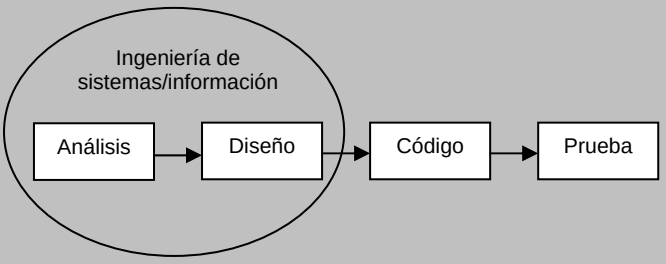


Figura 1.6 Modelo lineal secuencial

Ingeniería y modelado de sistemas/información.

Como el software siempre forma parte de un sistema más grande (o empresa), el trabajo comienza estableciendo requisitos de todos los elementos del sistema y asignando al software algún subgrupo de estos requisitos. Esta visión del sistema es esencial cuando el software se debe interconectar con otros elementos como hardware, personas y bases de datos. La ingeniería y el análisis de sistemas acompañan a los requisitos que se recogen en el nivel del sistema con una pequeña parte de análisis y de diseño. La ingeniería de información acompaña a los requisitos que se recogen en el nivel estratégico de empresa y en el nivel del área de negocio.

Análisis de los requisitos del software

El proceso de reunión de requisitos se intensifica y se centra especialmente en el software. Para comprender la naturaleza de él (los) programa(s) a construirse, el ingeniero (analista) de software debe comprender el dominio de la información del software así como la función requerida, comportamiento, rendimiento e interconexión. El cliente documenta y repasa los requisitos del software.

Diseño

El diseño del software es realmente un proceso de muchos pasos que se centra en cuatro atributos distintos de un programa: estructura de datos, arquitectura del software, representaciones de interfaz y detalle procedimental (algoritmo). El proceso de diseño traduce requisitos en una representación del software que se pueda evaluar por calidad antes de que comience la generación del código. Al igual que los requisitos, el diseño se documenta y se hace parte de la configuración del software.

Generación de código

El diseño se debe traducir en una forma legible por la máquina. El paso a código lleva a cabo esa tarea. Si se lleva a cabo el diseño de una forma detallada, la generación de código se realiza mecánicamente.

Pruebas

Una vez que se ha generado un código, comienzan las pruebas del programa. El proceso de pruebas se centra en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado, y en los procesos externos funcionales, es decir en la realización de las pruebas para la detección de los errores y el sentirse seguro de que la entrada definida produzca resultados reales de acuerdo con los resultados requeridos.

Mantenimiento

El software indudablemente sufrirá cambios después de ser entregado al cliente (una excepción posible es el software empotrado). Se producirán cambios porque se han encontrado errores, porque el software debe adaptarse para acoplarse a los cambios de su entorno externo (p. ej.: se requiere un cambio debido a un sistema operativo o dispositivo periférico nuevo), o porque le cliente requiere mejoras funcionales o de rendimiento. El mantenimiento vuelve a aplicar cada una de las fases precedentes a un programa ya existente y no a uno nuevo.

El modelo lineal secuencial es el paradigma más antiguo y más extensamente utilizado en la ingeniería de software. Sin embargo, la crítica al paradigma ha puesto en duda su eficacia. Entre los problemas que se encuentran algunas veces en el modelo lineal secuencial se incluyen:

1. Los proyectos reales raras veces siguen el modelo lineal secuencial que propone el modelo. Aunque el modelo lineal puede acoplar interacción, lo hace indirectamente. Como resultado, los cambios pueden causar confusión cuando el proyecto comienza.

2. A menudo es difícil que el cliente exponga explícitamente todos los requisitos. El modelo lineal secuencial lo requiere y tiene dificultades a la hora de acomodar la incertidumbre natural al comienzo de muchos proyectos.

3. El cliente debe tener paciencia. Una versión del trabajo del (los) programa(s) no estará disponible hasta que el proyecto este muy avanzado. Un grave error puede ser desastroso si no se detecta hasta que se revisa el programa.

4. Los responsables del desarrollo del software siempre se retrasan innecesariamente. En un interesante análisis de proyectos reales, Bradac dijo que la naturaleza lineal del ciclo de vida clásico lleva a estados de bloqueo en el que algunos miembros del equipo del proyecto deben de esperar a otros miembros del equipo para completar tareas dependientes. En efecto, el tiempo que se pasa esperando, puede sobrepasar el tiempo que se emplea en el trabajo productivo, los estados de bloqueo tienden a ser más importantes al principio y al final de un proceso lineal secuencial.

Cada uno de estos errores es real. Sin embargo, el paradigma del ciclo de vida clásico tiene un lugar definido e importante en el trabajo de la ingeniería del software. Proporciona una plantilla en la que se encuentran métodos para análisis, diseño, codificación, pruebas y mantenimiento. El ciclo de vida clásico, sigue siendo el modelo de proceso, más extensamente utilizado por la ingeniería del software. Pese a tener debilidades, es significativamente mejor que un enfoque hecho al azar para el desarrollo del software.

### 1.2.3 Modelo de prototipos

Un cliente a menudo define un conjunto de objetivos generales para el software, pero no identifica los requisitos detallados de entrada, procesamiento o salida. En otros casos, el responsable del desarrollo del software puede no estar seguro de la eficacia de un algoritmo, de la capacidad de adaptación de un sistema operativo, o de la forma en que debería tomarse la interacción hombre-máquina. En estas y en otras muchas situaciones, un paradigma de construcción de prototipos puede ofrecer el mejor enfoque.

El paradigma de construcción de prototipos (figura 1.7), comienza con la recolección de requisitos. El desarrollador y el cliente encuentran y definen los objetivos globales para el software, identifican los requisitos conocidos y las áreas del esquema en donde es obligatoria más definición. Entonces aparece un diseño rápido, el diseño rápido se centra en una representación de esos aspectos del software que serán visibles para el usuario/cliente (p.ej.: enfoques de entrada y formatos de salida). El diseño rápido lleva a la construcción de un prototipo. El prototipo lo evalúa el cliente/usuario y lo utiliza para refinar los requisitos del software a desarrollar. La interacción ocurre cuando el prototipo satisface las necesidades del cliente, a la vez que permite que el desarrollador comprenda mejor lo que se necesita hacer.

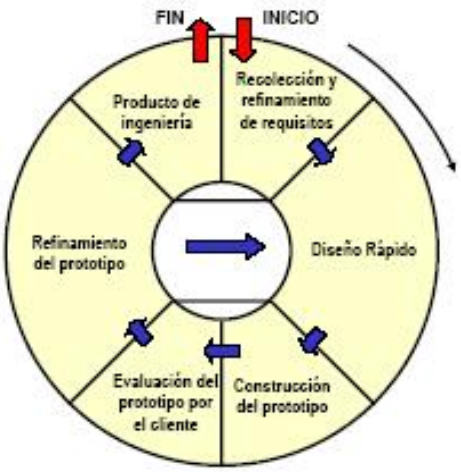


Figura 1.7 Representación gráfica del paradigma de prototipos

Lo ideal sería que el prototipo sirviera como un mecanismo para identificar los requisitos del software. Si se construye un prototipo de trabajo, el desarrollador intenta hacer uso de los fragmentos del programa ya existentes o aplica herramientas (p. ej.: generadores de informes, gestores de ventanas, etc.) que permiten generar rápidamente programas de trabajo.

Pero, ¿Qué hacemos con el prototipo una vez que ha servido para el propósito descrito anteriormente?

En la mayoría de los proyectos, el primer sistema construido apenas si se puede utilizar. Puede ser demasiado lento, demasiado grande, o torpe en su uso, o las 3 a la vez. No hay alternativa sino comenzar de nuevo y construir una versión rediseñada en la que se resuelvan estos problemas. Cuando se utiliza un concepto nuevo de sistema o una tecnología nueva, se tiene que construir un sistema que no sirva y se tenga que tirar. Porque incluso la mejor planificación no es omnisciente como para que esté perfecta la primera vez. Por tanto, la pregunta sobre la gestión no es si construir un

sistema piloto y tirarlo. Se tendrá que hacer. La única pregunta es si planificar de antemano construir un desechable o prometer entregárselo a los clientes.

El prototipo puede servir como primer sistema. Aunque éste puede ser una visión idealizada, es verdad que a los clientes y a los que desarrollan les gusta el paradigma de construcción de prototipos. A los usuarios les gusta el sistema real y a los que desarrollan les gusta construir algo inmediatamente. Sin embargo, la construcción de prototipos también puede ser problemática por las razones siguientes:

1. El cliente ve lo que parece ser una versión de trabajo del software, sin saber que con la prisa de hacer que funcione, no se ha tenido en cuenta la calidad del software global o la facilidad de mantenimiento a largo plazo. Cuando se informa de que el producto se debe construir una vez para que se puedan mantener los niveles altos de calidad, el cliente no lo entiende y pide que se apliquen algunos pequeños ajustes, para que se pueda hacer del prototipo un producto final. De forma demasiado frecuente la gestión de desarrollo del software es muy lenta.

2. EL desarrollador a menudo hace compromisos de implementación para hacer que el prototipo funcione rápidamente. Se puede utilizar un sistema operativo o lenguaje de programación inadecuado simplemente porque está disponible y porque es conocido; un algoritmo eficiente se puede implementar simplemente para demostrar la capacidad. Después de algún tiempo, el desarrollador debe de familiarizarse con estas selecciones y olvidarse de las razones por las que son inadecuadas. La selección menos ideal ahora es parte del sistema.

Aunque pueden surgir problemas, la construcción de prototipos puede ser un paradigma efectivo para la ingeniería del software. La clave es definir las reglas del juego al comienzo; es decir, el cliente y el desarrollador se deben de poner de acuerdo en que el prototipo que se construya para servir como un mecanismo de definición de requisitos. Entonces se descarta (al menos en parte), y se realiza la ingeniería del software con una visión hacia la calidad y la facilidad de mantenimiento.

### 1.2.4 Modelo de Desarrollo Rápido de Aplicaciones DRA

El Desarrollo Rápido de Aplicaciones DRA o RAD (del inglés *Rapid Application Development*) es un modelo del desarrollo del software lineal secuencial, que enfatiza un ciclo de desarrollo extremadamente corto. El modelo DRA es una adaptación a alta velocidad del modelo lineal secuencial en el que se logra el desarrollo rápido utilizando un enfoque de construcción basado en componentes. Si se comprenden bien los requisitos y se limita el ámbito del proyecto, el proceso DRA permite al equipo de desarrollo crear un sistema completamente funcional dentro de periodos cortos de tiempo (p.ej.: de 60 a 90 días). Cuando se utiliza principalmente para aplicaciones de sistemas de información, el enfoque DRA comprende las siguientes fases.

Modelado de gestión

El flujo de información entre las funciones de gestión se modela de forma que responda a las siguientes preguntas; ¿Qué información conduce al proceso de

gestión?,¿Qué información se genera?,¿Quién la genera?,¿A dónde va la información?,¿Quién la procesa?.

Modelado de datos

El flujo de información definido como parte de la fase de modelado de gestión se refina como un conjunto de objetos de datos necesarios para apoyar la empresa. Se definen las características (llamadas atributos) de cada uno de los objetos y las relaciones entre estos objetos.

Modelado del proceso

Los objetos de datos definidos en la fase de modelado de datos quedan transformados para lograr el flujo de información necesario para implementar una función de gestión. Las descripciones del proceso se crean para añadir, modificar, suprimir o recuperar un objeto de datos.

Generación de aplicaciones

El DRA asume la utilización de técnicas de cuarta generación. En lugar de crear el software con lenguajes de programación de tercera generación, el proceso DRA trabaja para volver a utilizar componentes de programas ya existentes (cuando es posible) o crear componentes reutilizables (cuando sea necesario). En todos los casos se utilizan herramientas automáticas para facilitar la construcción del software.

Pruebas y entrega

Como el proceso DRA enfatiza la reutilización, ya se han comprobado muchos de los componentes de los programas. Esto reduce tiempo de pruebas. Sin embargo, se deben probar todos los componentes nuevos y se deben ejercitar todas las interfaces a fondo.

El modelo de proceso DRA se ilustra en la figura 1.8, obviamente las limitaciones de tiempo impuestas por un proyecto DRA demandan ámbito en escalas. Si una aplicación de gestión puede modularse de forma que permita completarse cada una de las funciones principales en menos de 3 meses (utilizando el enfoque descrito anteriormente), es un candidato del DRA. Cada una de las funciones puede ser afrontada por un equipo DRA diferente y ser integrado en un solo conjunto.

Al igual que todos los modelos de proceso, el enfoque DRA tiene inconvenientes:

- Para proyectos grandes aunque por escalas el DRA requiere recursos humanos suficientes como para crear el número correcto de equipos DRA.
- DRA requiere clientes y desarrolladores comprometidos en las rápidas actividades necesarias para completar un sistema en un marco de tiempo abreviado. Si no hay compromiso, por ninguna de las partes constituyentes, los proyectos DRA fracasarán.

No todos los tipos de aplicaciones son apropiados para DRA. Si un sistema no se puede modularizar adecuadamente, la construcción de los componentes necesarios para DRA será problemático. Si está en juego el alto rendimiento, y se va a conseguir el rendimiento convirtiendo interfaces en componentes de sistemas, el enfoque DRA puede que no funcione, DRA no es adecuado cuando los riesgos técnicos son altos. Esto ocurre cuando una nueva aplicación hace uso de tecnologías nuevas, o cuando el nuevo software requiere un alto grado de interoperatividad con programas de

computadoras ya existentes. En la figura 1.8 se muestra de manera esquemática el modelo DRA.

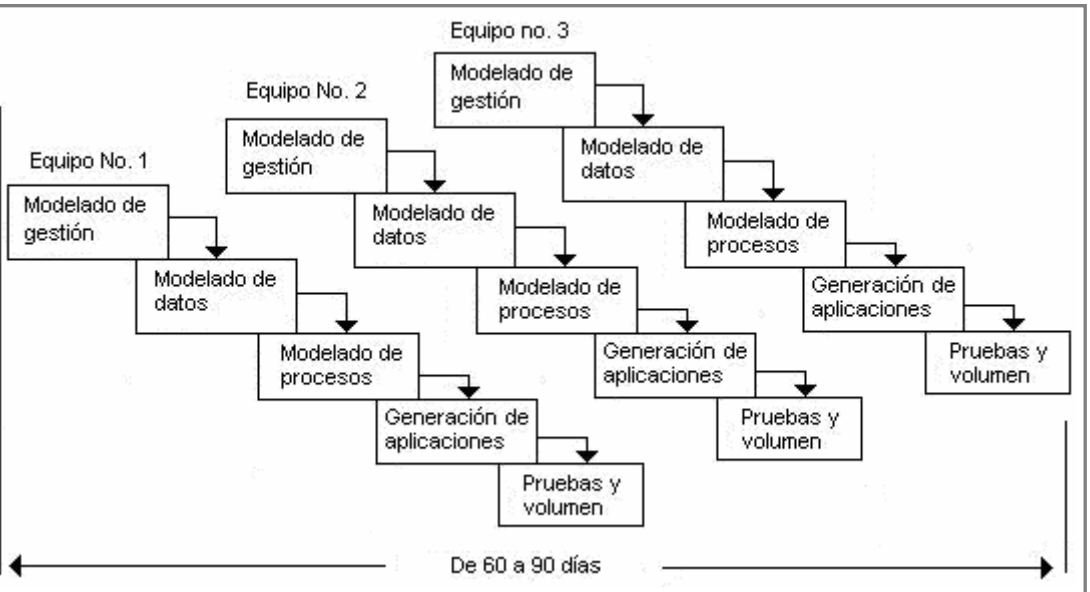


Figura 1.8 Modelo DRA

DRA enfatiza la reutilización de componentes de programas como piedra angular del desarrollo de software.

### 1.2.5 Modelo de procesos evolutivos de software

Se reconoce que el software al igual que todos los sistemas complejos, evoluciona con el tiempo. Los requisitos de gestión y de productos a menudo cambian conforme a que el desarrollo proceda haciendo que el camino que llega al producto final no sea real; las estrictas fechas tope del mercado hacen que sea imposible finalizar un producto completo por lo que se debe introducir una versión limitada para cumplir la presión competitiva y de gestión; se comprende perfectamente el conjunto de requisitos de productos centrales o del sistema, pero todavía se tienen que definir los detalles de extensiones del producto o sistema. En éstas y otras situaciones similares, los ingenieros del software necesitan un modelo de proceso que se haya diseñado explícitamente para acomodarse a un producto que evolucione con el tiempo.

El modelo lineal secuencial se diseña para el desarrollo en línea recta. En esencia este enfoque en cascada asume que se va a entregar un sistema completo una vez que la secuencia lineal se haya finalizado. El modelo de construcción de prototipos se diseña para ayudar al cliente (o al que desarrolla) a comprender los requisitos. En general no se diseña para entregar un sistema de producción. En ninguno de los paradigmas de ingeniería de software se tiene en cuenta la naturaleza evolutiva del software.

Los modelos evolutivos son iterativos. Se caracterizan por la forma en que permiten a los ingenieros del software desarrollar versiones cada vez más completas del software.

### 1.2.6 Modelo incremental

El modelo incremental combina elementos del modelo lineal secuencial (aplicados repetitivamente) con la filosofía interactiva de construcción de prototipos. Como

muestra la figura 1.9, el modelo incremental aplica secuencias lineales de forma sorprendente de la misma forma que progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software. Por ejemplo, el software de tratamiento de textos desarrollado con el paradigma incremental podría extraer funciones de gestión de archivos básicos y de producción de documentos en el primer incremento; funciones de edición más sofisticadas y de producción de documentos en el segundo incremento; corrección ortográfica y gramatical en el tercero; y una función avanzada de esquema de página en el cuarto. Se deberá tener en cuenta que el flujo del proceso de cualquier incremento puede incorporar el paradigma de construcción de prototipos.

Cuando se utiliza un modelo incremental, el primer incremento a menudo es un producto esencial (núcleo), es decir, se afrontan requisitos básicos, pero muchas funciones suplementarias (algunas conocidas, otras no) quedan sin extraer. El cliente utiliza el producto central (o sufre la revisión detallada). Como un resultado de utilización y/o evaluación, se desarrolla un plan para el incremento siguiente. El plan afronta la modificación del producto central a fin de cumplir mejor las necesidades del cliente y la entrega de funciones, y características adicionales. Este proceso se repite siguiendo la entrega de cada incremento, hasta que se elabore el producto completo.

El modelo del proceso incremental, como la construcción de prototipos y otros enfoques evolutivos, es interactivo por naturaleza. Pero a diferencia de la producción de prototipos, el modelo incremental se centra en la entrega de un producto operacional con cada incremento. Los primeros incrementos son versiones desmontadas del producto final, pero proporcionan la capacidad que sirve al usuario y también proporcionan una plataforma para la evaluación por parte del usuario.

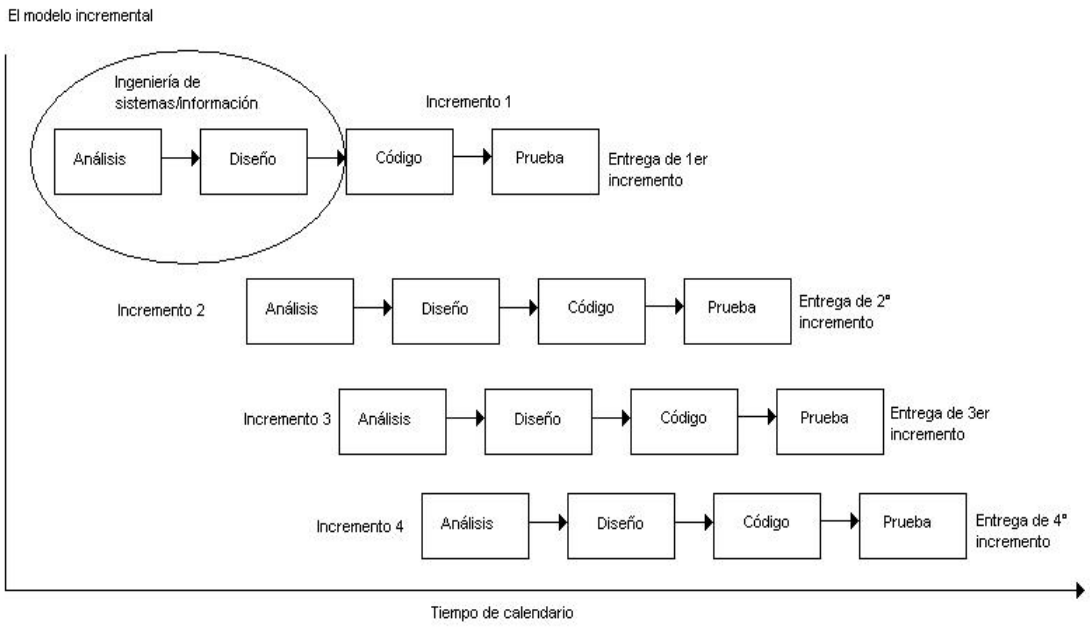


Figura 1.9 Modelo evolutivo

El desarrollo incremental es particularmente útil cuando la dotación de personal no está disponible para una implementación completa en cuanto a la fecha límite de gestión que se haya establecido para el proyecto. Los primeros incrementos se

pueden implementar con menos personas. Si el producto central es bien recibido, se puede añadir más personal (si se requiere) para implementar el incremento siguiente. Además, los incrementos se pueden planear para gestionar riesgos técnicos. Por ejemplo, un sistema importante podría requerir la disponibilidad de hardware nuevo que esté bajo desarrollo y cuya fecha de entrega no sea cierta. Podría ser posible planificar primeros incrementos de forma que se evite la utilización de este hardware, y así se permita entregar la funcionalidad parcial a usuarios finales, sin un retraso exagerado.

### 1.2.7 Modelo en espiral

El modelo en espiral, es un modelo de proceso de software evolutivo que acompaña a la naturaleza interactiva de construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial. Se proporciona el potencial para el desarrollo rápido de versiones incrementales del software. En el modelo espiral, el software se desarrolla en una serie de versiones incrementales. Durante las primeras iteraciones, la versión incremental podría ser un modelo en papel o un prototipo. Durante las últimas iteraciones, se producen versiones cada vez más complejas de ingeniería del sistema.

El modelo en espiral se divide en un número de actividades estructurales también llamadas regiones de tareas. Generalmente existen entre tres y seis regiones de tareas. La figura 1.10 representa un modelo en espiral que contiene seis regiones de tareas:

- Comunicación con el cliente - Las áreas requeridas para establecer la comunicación entre el desarrollador y el cliente.
- Planificación - Las tareas requeridas para definir recursos, el tiempo y otras informaciones relacionadas con el proyecto.
- Ingeniería - Las tareas requeridas para evaluar riesgos técnicos y de gestión.
- Construcción y aplicación - Las tareas requeridas para construir una o más representaciones de la aplicación.
- Evaluación del cliente - Las tareas requeridas para construir, probar, instalar y proporcionar soporte al usuario (p. ej.: documentación y práctica).

Cada una de las regiones está poblada por una serie de tareas que se adaptan a las características del proyecto que va a emprenderse. Para proyectos pequeños, el número de tareas y su formalidad es bajo. Para proyectos mayores y más críticos, cada región contiene tareas que se definen para lograr un nivel más alto de formalidad. En todos los casos, se aplican las actividades de protección (p. ej.: gestión de configuración del software y garantía de calidad del software).

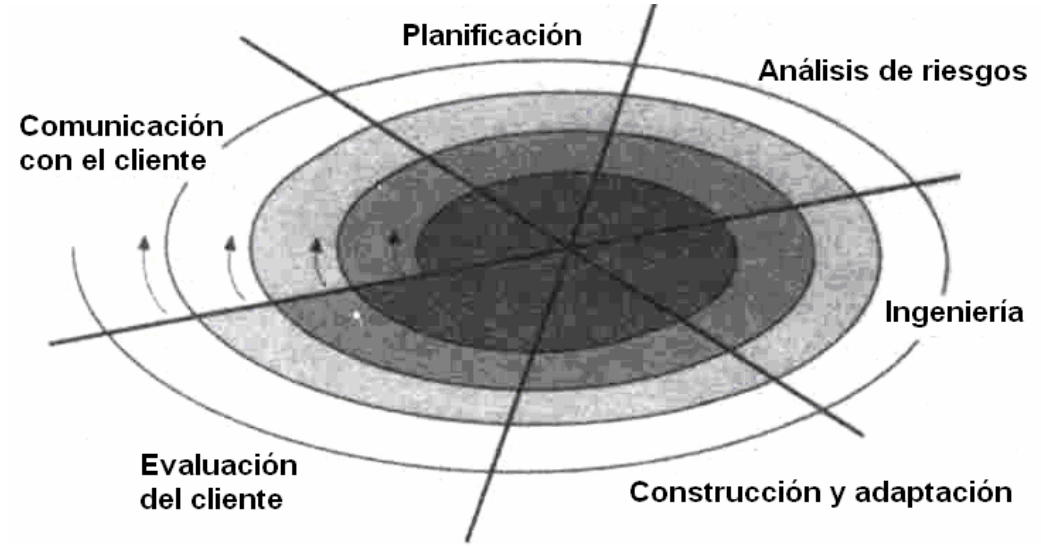


Figura 1.10 Modelo en especial típico

Cuando empieza este proceso evolutivo, el equipo de ingeniería del software, gira alrededor de la espiral en la dirección de las manecillas del reloj, comenzando por el centro. El primer circuito de la espiral produce el desarrollo de una especificación de productos; los pasos siguientes en la espiral se podrían utilizar para desarrollar un prototipo y progresivamente versiones más sofisticadas del software. Cada paso de la región de planificación produce ajustes en el plan del proyecto. El coste y la planificación se ajustan según la reacción ante la evaluación del cliente. Además el gestor del proyecto ajusta el número planificado de iteraciones requeridas para completar el software.

A diferencia del modelo del proceso clásico que termina cuando se entrega el software, el modelo en espiral puede adaptarse y aplicarse a lo largo de la vida del software de computadora. En la figura 1.11 se define un eje de punto de entrada en el proyecto. Cada uno de los cubos situados a lo largo del eje representa el punto de arranque para un tipo diferente de proyecto. Un proyecto de desarrollo de conceptos comienza en el centro de la espiral y continuará (aparecen múltiples iteraciones a lo largo de la espiral que limita la región sombreada central) hasta que se completa el desarrollo del concepto. Si el concepto se va a desarrollar dentro de un producto real, el proceso procede a través del cubo siguiente (punto de entrada del producto nuevo) y se inicia un nuevo proyecto de desarrollo. El producto nuevo evolucionará a través de iteraciones alrededor de la espiral siguiendo el camino que limita la región algo más brillante que el centro. Un flujo de proceso similar aparece para otros tipos de proyectos.

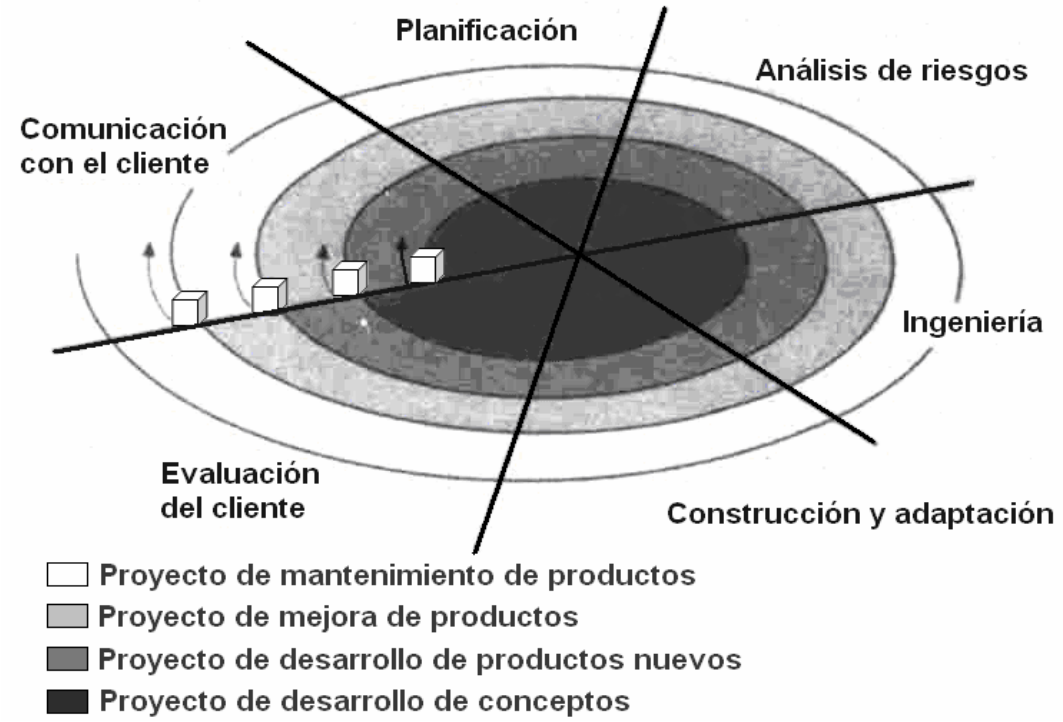


Figura 1.11 Modelo en espiral adaptado para el ciclo de vida clásico completo

En esencia, la espiral, cuando se caracteriza de esta forma, permanece operativa hasta que el software se retira, hay veces que el proceso está inactivo, pero siempre que se inicie un cambio, el proceso arranca en el punto de entrada adecuado (p. ej.: mejora del producto).

El modelo en espiral es un enfoque realista del desarrollo de sistemas y de software a gran escala. Como el software evoluciona a medida que progresa el proceso, el desarrollador y el cliente comprenden y reaccionan mejor ante riesgos en cada uno de los niveles evolutivos. El modelo en espiral utiliza la construcción de prototipos como mecanismo de reducción de riesgos, pero lo que es más importante, permite a quien lo desarrolla aplicar el enfoque de construcción de prototipos en cualquier etapa de evolución del producto. Mantiene el enfoque sistemático de los pasos sugeridos por el ciclo de vida clásico, pero lo incorpora al marco de trabajo interactivo que refleja de forma más realista el mundo real. El modelo en espiral demanda una consideración directa de los riesgos técnicos en todas las etapas del proyecto, y si se aplica adecuadamente, debe reducir los riesgos antes de que se conviertan en problemáticos.

Pero al igual que otros paradigmas, el modelo en espiral no es una panacea. Puede resultar difícil convencer a grandes clientes (particularmente en situaciones bajo contrato) de que el enfoque evolutivo es controlable. Requiere una considerable habilidad para la evaluación del riesgo, y cuenta con esta habilidad para el éxito. Si un riesgo importante no es descubierto y gestionado, indudablemente surgirán problemas. Finalmente el modelo en sí mismo es relativamente nuevo y no se ha utilizado tanto como los paradigmas lineales secuenciales o de construcción de prototipos. Todavía tendrán que pasar muchos años antes de que se determine con absoluta certeza la eficacia de este nuevo e importante paradigma.

### 1.2.8 Modelo de ensamblaje de componentes

Las tecnologías de objetos proporcionan el marco de trabajo técnico para un modelo de proceso basado en componentes para la ingeniería del software. El paradigma de orientación a objetos enfatiza la creación de clases que encapsulan tanto los datos como los algoritmos que se utilizan para manejar los datos. Si se diseñan y se implementan adecuadamente, las clases orientadas a objetos son reutilizables por las diferentes aplicaciones y arquitecturas de sistemas basados en la computadora.
El modelo de ensamblaje de componentes (fig. 1.12) incorpora muchas de las características del modelo en espiral. Es evolutivo por naturaleza y exige un enfoque interactivo para la creación del software. Sin embargo el modelo ensamblador de componente configura aplicaciones desde componentes preparados de software (algunas veces llamados clases).

La actividad de la ingeniería comienza con la identificación de clases candidatas. Esto se lleva a cabo examinando los datos que se van a manejar por parte de la aplicación y el algoritmo que se va a aplicar para conseguir el tratamiento. Los datos y los algoritmos correspondientes se almacenan en una clase.

Las clases (llamadas componentes en la figura 1.12) creadas en los proyectos de ingeniería de software anteriores se almacenan en una biblioteca de clases o depósito. Una vez identificadas las clases candidatas, la biblioteca de clases se examina para determinar si estas clases ya existen. En caso de que así fuera, se extraen de la biblioteca y se vuelven a utilizar. Si una clase candidata no existe en la biblioteca, se aplican los métodos orientados a objetos. Se compone así la primera interacción de la aplicación a construirse, mediante las clases extraídas de la biblioteca y las clases nuevas construidas para cumplir las necesidades únicas de la aplicación. El flujo del proceso vuelve a la espiral y volverá a introducir por último la iteración ensambladora de componentes a través de la actividad de ingeniería.

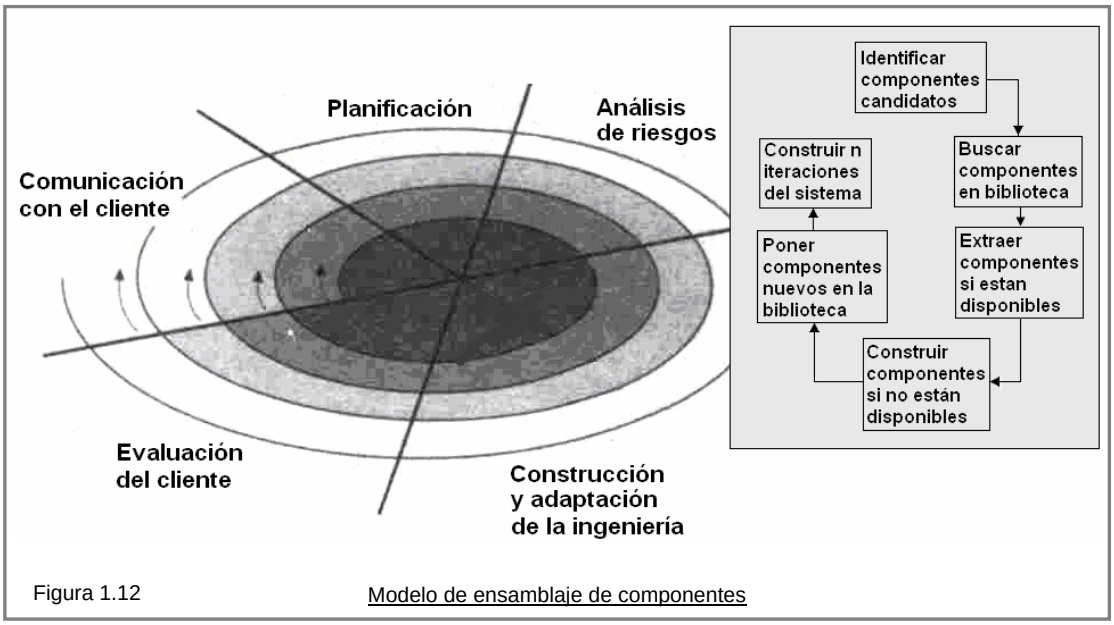


Figura 1.12 Modelo de ensamblaje de componentes

El modelo ensamblador de componentes lleva a la reutilización del software, y la reutilización proporciona beneficios a los ingenieros del software. Según estudios de reutilización, QSM Associates. Inc. informa que el ensamblaje de componentes lleva a una reducción del 70% del tiempo del ciclo de desarrollo, un 84% del coste del

proyecto y un índice de productividad del 26.2 comparado con la norma de la industria del 16.9. Aunque estos resultados están en función de la robustez de la biblioteca de componentes, no hay duda de que el ensamblaje de componentes proporciona ventajas significativas para los ingenieros del software.

### 1.2.9 Técnicas de cuarta generación T4G

El termino de técnicas de cuarta generación (T4G) Abarca un amplio espectro de herramientas de software que tienen algo en común: todas facilitan al ingeniero del software la especificación de algunas características del software a alto nivel. Luego, la herramienta genera automáticamente el código fuente basándose en la especificación del técnico. Cada vez parece más evidente que cuanto mayor sea el nivel en la que se especifique el software, mas rápido se podrá conseguir el programa.

El paradigma T4G para la ingeniería del software se orienta hacia la posibilidad de especificar el software o notaciones graficas que describan el problema que hay que resolver en términos que los entienda el cliente.

Actualmente, un entorno para el desarrollo del software que soporte el paradigma T4G puede incluir todas o algunas de las siguientes herramientas:

- Lenguajes no procedimentales de consulta a bases de datos.
- Generación de informes.
- Manejo de datos.
- Interacción y definición de pantallas.
- Generación de códigos.
- Capacidades gráficas de alto nivel.
- Capacidades de hoja de cálculo.

Inicialmente estas herramientas eran utilizadas pero solo para aplicaciones muy especificas, y ahora la T4G se ha extendido a todas las categorías de aplicaciones del software.

Las herramientas T4G generan automáticamente el código fuente basándose en el análisis y el diseño.

T4G comienza con el paso de reunión de requisitos; el dialogo cliente-desarrollador descrito por los otros paradigmas sigue siendo una parte esencial del enfoque T4G.
El uso de T4G sin diseño (para grandes proyectos) causara las mismas dificultades (poca calidad, mantenimiento pobre, mala aceptación por el cliente) que se encuentran cuando se desarrolla software mediante los enfoques convencionales.

La implementación mediante L4G permite, al que desarrolla el software centrarse en la implementación del código fuente.

Para transformar una implementación de T4g en un producto, el que lo desarrolla debe dirigir una prueba completa, desarrollar con sentido una documentación y ejecutar el resto de las actividades de integración que son también requeridas por otros paradigmas de ingeniería de software. Además, el software desarrollado con T4G debe de ser construido de forma que facilite la realización del mantenimiento de forma expedita

Defensores y detractores

Los defensores aducen reducciones drásticas en el tiempo de desarrollo del software y una mejora significativa en la productividad de la gente que construye el software.

Los detractores aducen que las herramientas actuales de T4G no son más fáciles de utilizar que los lenguajes de programación; que el código fuente producido por tales herramientas es "ineficiente" y que el mantenimiento de grandes sistemas de software desarrollados mediante T4G, es cuestionable.

Hay un algún merito en lo que se refiere a indicaciones de ambos lados y es posible el estado actual de los enfoques de T4G:

1. El uso de T4G es un enfoque viable para muchas de las diferentes áreas de aplicación. Junto con las herramientas de la ingeniería de software asistida por computadora (CASE) y los generadores de código, T4G ofrece una solución fiable a muchos problemas de el software.
2. Los datos recogidos en compañías que usan T4G parecen indicar que el tiempo requerido para producir software se reduce mucho para aplicaciones pequeñas y de tamaño medio, ya que la cantidad de análisis y diseño para las aplicaciones pequeñas también se reducen.
3. Sin embargo, el uso de T4G para grandes trabajos de desarrollo de software exigen el mismo tiempo o más de análisis, diseño y prueba (actividades de ingeniería de software), para lograr un ahorro sustancial de tiempo que puede conseguirse mediante la eliminación de la codificación.

Ventajas

Reducción en tiempo de desarrollo.

Desventajas

- Código ineficiente.
- No más fáciles de usar que L3G.
- Mantenimiento cuestionable.

Las técnicas de la cuarta generación ya se han convertido en una parte de suma importancia para el desarrollo del software. Cuando se combinan con enfoques de ensamblaje de componentes, el paradigma de la cuarta generación se puede convertir en el enfoque dominante hacia el desarrollo del software.

Pero sin embargo aunque se usen técnicas de cuarta generación se debe hacer análisis, diseño y pruebas con el propósito de evitar la mala calidad, mantenimiento pobre, baja aceptación por el cliente y en consecuencia fracaso de los proyectos.

# Capítulo II

# Herramientas y fundamentos teóricos

## 2.1 Entorno UNIX

Unix es un sistema operativo portable, multitarea y multiusuario; desarrollado, a principios de 1969 por un grupo de empleados de los laboratorios Bell de AT&T, entre los que figuran Ken Thompson, Dennis Ritchie y Douglas McIlroy.

Hasta 2007, el propietario de la marca UNIX® es The Open Group, un consorcio de normalización industrial. Sólo los sistemas que cumplen a cabalidad y se encuentran certificados por la especificación “Single UNIX Specification” o “Especificación Única de Unix”, pueden ser denominados "UNIX®" (otros reciben la denominación "similar a un sistema Unix" o "similar a Unix").

Después de que los laboratorios Bell abandonaran el proyecto MULTICS, Ken Thompson, uno de los programadores del proyecto MULTICS, siguió trabajando para la computadora GE-635 y escribió un juego llamado Space Travel (Viaje espacial). Sin embargo, descubrió que el juego era lento en la máquina de General Electric y resultaba realmente caro, algo así como 75 dólares de americanos por cada partida.

Derivado de esto, Thompson reescribió el programa, con ayuda de Dennis Ritchie, en lenguaje ensamblador, para que se ejecutase en una computadora DEC PDP-7. Esta experiencia, junto al trabajo que desarrolló para el proyecto Multics, condujo a Thompson a iniciar la creación de un nuevo sistema operativo para la DEC PDP-7 . Thompson y Ritchie lideraron un grupo de programadores, entre ellos a Rudd Canaday, en los laboratorios Bell, para desarrollar tanto el sistema de archivos como el sistema operativo multitarea en sí. A lo anterior, agregaron un intérprete de órdenes (o intérprete de comandos) y un pequeño conjunto de programas. El proyecto fue bautizado UNICS, como acrónimo Uniplexed Information and Computing System, pues sólo prestaba servicios a dos usuarios. La autoría de estas siglas se le atribuye a Brian Kernighan, como un juego palabras de Multics, en el que se hacia alusión a que UNICS era un sistema MULTICS castrado, dado que además eunuchs, en inglés, es un homófono de UNICS.

Debido a esto se cambió el nombre de UNICS a Unix, dando origen al legado de este poderoso sistema operativo que llega hasta nuestros días.

En 1972 se tomó la decisión de escribir nuevamente Unix, pero esta vez en el lenguaje de programación C. Este cambio significaba que Unix podría ser fácilmente modificado para funcionar en otras computadoras (de esta manera, se volvía portable) y así otras variaciones podían ser desarrolladas por otros programadores. Ahora, el código era más conciso y compacto, lo que se tradujo en un aumento en la velocidad de desarrollo de Unix. AT&T puso a Unix a disposición de universidades y compañías, también al gobierno de los Estados Unidos, a través de licencias. Una de estas licencias fue otorgada al Departamento de Computación de la Universidad de California, con sede en Berkeley. En 1975 esta institución desarrolló y publicó su propia versión de Unix, conocida como Distribución de Software de Berkeley o BSD (por sus iniciales en inglés *Berkeley Software Distribution*), que se convirtió en una fuerte competencia para la familia Unix de AT&T.

En 1993, la compañía Novell adquirió la división Unix Systems Laboratories de AT&T junto con su propiedad intelectual. Esto ocurrió en un momento delicado en el que Unix Systems Laboratories disputaba una demanda en los tribunales contra BSD por infracción de los derechos de copyright, revelación de secretos y violación de marca de mercado.

BSD no solamente ganó el juicio sino que cambiaron los papeles completamente, descubriendo que grandes porciones del código de BSD habían sido copiadas ilegalmente en Unix System V. En realidad, la propiedad intelectual de Novell (recién adquirida de Unix Systems Laboratories) se reducía a unos pocos archivos fuente. La correspondiente contra-demanda acabó en un acuerdo extrajudicial cuyos términos permanecen bajo secreto a petición de Novell.

Aproximadamente por esas mismas fechas, un estudiante de ciencias de la computación, llamado Linus Torvalds desarrolló un núcleo ó kernel para computadoras de arquitectura Intel x86 que emulaba las funcionalidades de Unix y que fue lanzado como código abierto en 1991, bajo el nombre de Linux. En 1992, el kernel de Linux fue combinado con los programas desarrollados por el proyecto GNU, dando como resultado el sistema operativo GNU/Linux.

Características

Unix es un sistema operativo multiusuario y multitarea que se amolda a un modelo típico de capas, de forma que cada capa únicamente puede comunicarse con las capas que se hallan en los niveles inmediatamente inferior y superior. Estas capas son desde la capa interior a la exterior, las siguientes:

- El Hardware.
- El Núcleo o Kernel.
- El Intérprete de comandos o Shell.
- El Sistema de Archivos.
- Los Programas de Usuario.

El Kernel es un programa escrito casi en su totalidad en lenguaje C, con excepción de una parte del manejo de interrupciones, expresada en el lenguaje ensamblador del procesador en el que opera.

El kernel interactúa directamente con el hardware y proporciona una serie de servicios comunes a los programas de las capas superiores, de forma que las peculiaridades del hardware permanecen ocultas. Como los programas son independientes del hardware, es fácil mover programas entre sistemas Unix que se ejecutan en hardware diferente.

Las características generales de los sistemas operativos Unix son:

- Es interactivo: permite el diálogo entre el usuario y el computador. El sistema acepta órdenes, las ejecuta y se dispone a esperar otras nuevas.
- Es multitarea: permite que se puedan ejecutar varios procesos al mismo tiempo compartiendo el uso del procesador.
- Es multiusuario: permite a varios usuarios compartir los recursos del computador simultáneamente.
- Es portable: es un sistema independiente del procesador y del equipo, esto se debe a que en su mayoría esta escrito en “C”, por lo cual puede ser portado a cualquier computador.
- Posee distintos niveles de seguridad, incluyendo claves de ingreso al sistema; y permisos de acceso a los archivos y directorios. Contiene un potente lenguaje de programación de comando (SHELL) lo cual permite a los usuarios la creación de sus propios comandos.
- Estructura jerárquica de archivos.
- Permite trabajar en modo background.

- Maneja procesos diferidos, procesos que se ejecutan a determinado horario.

Sistema de Archivos de Unix

El sistema de archivos de Unix está basado en un modelo arborescente y recursivo, en el cual los nodos pueden ser tanto archivos como directorios, y estos últimos pueden contener a su vez directorios o subdirectorios. Debido a esta filosofía, se maneja al sistema con muy pocas órdenes, que permiten una gran gama de posibilidades. Todo archivo de Unix está controlado por múltiples niveles de protección, que especifican los permisos de acceso al mismo. La diferencia que existe entre un archivo de datos, un programa, un manejador de entrada/salida o una instrucción ejecutable se refleja en estos parámetros, de modo que el sistema operativo adquiere características de coherencia y elegancia que lo distinguen.

### 2.1.1 Orígenes e historia

MULTICS

Para hablar de Unix es por fuerza necesario hablar primero de MULTICS (por sus iniciales en inglés *Multiplexed Information and Computing Service*) el cual fue uno de los primeros sistemas operativos de tiempo compartido y tuvo una gran influencia en el desarrollo de los posteriores sistemas operativos.

Los planes iniciales y el desarrollo de Multics comenzaron en 1964. Originalmente era un proyecto cooperativo liderado por Fernando J. Corbató del MIT, con General Electric y los laboratorios Bell. Los laboratorios Bell abandonaron el proyecto en abril de 1969, y en 1970 el negocio de computación de General Electric, incluyendo Multics, fue adquirido por Honeywell.

Multics fue concebido como un producto comercial por General Electric, y alcanzó este logro para Honeywell, pero no tuvo gran éxito. Sin embargo, tuvo un gran impacto en el campo de la computación gracias a sus muchas nuevas y valiosas ideas. Aunque en su época recibió muchas críticas, la historia ha demostrado que eran infundadas.

Un gran número de características intentaban proporcionar alta disponibilidad, de manera que el servicio de computación igualase a los servicios de telefonía y a la red eléctrica. Para alcanzar este logro, se utilizó una estructura modular tanto en el software como en el hardware, y el sistema podía crecer simplemente añadiendo más recursos - poder de computación, memoria principal, almacenamiento de disco, etcétera. Las listas de control de acceso sobre cada archivo proporcionaban un método flexible para compartir información, además de la privacidad requerida. Contenía diferentes mecanismos estándar para permitir a los ingenieros analizar el rendimiento del sistema, además de varios mecanismos para la optimización del rendimiento.

Ideas nuevas

Multics fue un de los primeros sistemas operativos que implementó un único nivel de almacenamiento para el acceso a los datos, desechando la clara distinción entre los archivos (llamados *segmentos* en Multics) y los procesos en memoria. La memoria de un proceso consistía solamente en segmentos que estaban mapeados en su espacio de direcciones; para leer o escribir en ellos, el proceso simplemente utilizaba instrucciones normales del CPU, y el sistema operativo tenía cuidado de asegurarse que todas las modificaciones fueran guardadas en disco. En la terminología POSIX,

era como si cada archivo fuese mapeado; sin embargo, en Multics no existía el concepto de *memoria de proceso*, separado del de la memoria utilizada para mantener mapeos sobre los archivos, como hace Unix. Toda la memoria del sistema formaba parte de algún segmento, que aparecía en el sistema de archivos; incluida la memoria temporal del proceso, la pila del kernel, etc.

Esto nos conduce a la segunda gran idea de Multics, enlace dinámico, mediante el que un proceso en ejecución puede solicitar que otros segmentos se añadan a su espacio de direcciones, estos segmentos pueden incluir código que puede ser ejecutado.

Un enlace dinámico es aquel en el cual una biblioteca de código es enlazada cuando un determinado programa se ejecuta (en oposición a un enlace estático, que se produce en tiempo de compilación). La ventaja de este tipo de enlace es que el programa es más liviano, y que evita la duplicación de código (por ejemplo, cuando dos programas enlazan con la misma biblioteca, se necesita sólo una copia).

Muchos programas tienen procedimientos a los que no llaman, salvo en circunstancias excepcionales. Aquí, después del ensamblaje, podemos enlazar cada procedimiento en el momento en que es llamado.

Con esta característica disponible, las aplicaciones automáticamente utilizaban la última versión de cualquier rutina externa que llamaban, estas rutinas estaban en otros segmentos, que se enlazaban dinámicamente sólo cuando un proceso intentaba ejecutarlas. Como diferentes procesos, pertenecientes a diferentes usuarios, podían utilizar diferentes reglas de búsqueda, diferentes usuarios podían automáticamente acabar utilizando diferentes versiones de las rutinas externas. Igualmente importante, con la configuración adecuada de las características de seguridad de Multics, el código en el otro segmento podía ganar acceso a las estructuras de datos mantenidas en un proceso diferente.

De este modo, para interactuar con una aplicación ejecutándose en parte como un demonio (en otro proceso) un proceso de usuario simplemente realiza una llamada normal a un procedimiento, hacia un segmento de código con el que se enlazó dinámicamente (un segmento de código que implementa alguna operación asociada con este demonio). El código en este segmento puede entonces modificar los datos mantenidos y utilizados por el demonio. Cuando la acción necesaria para comenzar la solicitud se completa, una simple instrucción de retorno del proceso retorna el control del proceso de usuario al código del usuario.

Debe notarse que estas dos ideas, se utilizan todavía en la mayoría de los sistemas operativos, independientemente del rápido y enorme avance realizado en el campo de la computación desde la década de 1960.

Multics también soportaba una reconfiguración en línea muy agresiva; las CPUs, los bancos de memoria, unidades de disco, etcétera podían ser añadidas y quitadas mientras el sistema continuaba funcionando; entrando en servicio, o eliminándose cuando fuera necesario. (De hecho, era una práctica común en el sistema del MIT, donde se realizó la mayoría de los desarrollos de software iniciales, dividir el sistema; que era un sistema multiprocesador; en dos sistemas separados, al eliminar suficiente componentes para formar un segundo sistema, dejando el resto ejecutando para usuarios que tenían todavía una sesión abierta. Las comprobaciones sobre el software desarrollado podían realizarse en la segunda máquina; cuando las pruebas concluían, los componentes del segundo sistema se añadían de nuevo en el sistema principal, sin

tener que apagarlo en ningún momento). Fue sin duda uno de los primeros sistemas multiprocesador y hotswap del mundo.

Multics también fue notable por su innovador énfasis en su diseño para obtener seguridad informática, y Multics fue posiblemente el primer sistema operativo diseñado como un sistema seguro desde su inicio. Independientemente de esto, las primeras versiones de Multics se mostraron vulnerables, no una vez, sino de forma repetida.

Esto condujo a más trabajo para mejorar la seguridad que trazó las técnicas de seguridad modernas, e hizo los sistemas más seguros.

Además de haber sido el primer sistema operativo que proporcionó un sistema de archivos jerárquico, los nombres de los archivos casi podían tener una longitud y sintaxis arbitraria; dado un archivo o directorio este podía tener varios nombres (típicamente uno largo y otro corto); también se soportaban enlaces simbólicos entre directorios.

Fue el primero en utilizar el ahora estándar concepto de tener una pila por proceso en el núcleo, con una pila separada para cada anillo de seguridad.

En retrospectiva

Es más que sorprendente descubrir que el kernel de este poderoso mainframe multiprocesador *computing utility*, tan criticado en sus días por ser demasiado grande y complejo, ocupaba 135 Kb de código. El primer GE-645 del MIT tuvo 512K palabras de memoria (2 MB), de manera que el kernel sólo utilizaba una pequeña porción de la memoria principal de Multics.

Medido de otra forma, el sistema completo, incluyendo no sólo el sistema operativo, sino también el complejo compilador de PL/I, los comandos de usuario, y las bibliotecas de subrutinas, contenía unos 1500 módulos de código fuente. Cada uno con una longitud media de unas 200 líneas de código fuente, y compilaban para producir aproximadamente 4,5 MB de código de procedimientos, que aunque hoy parezca pequeño era muy grande para los estándares de la época.

Los compiladores de Multics generalmente optimizaban la densidad de código antes que la velocidad de ejecución, por ejemplo se utilizaban pequeñas subrutinas llamadas operadores para secuencias de código frecuentes. Realizar comparaciones directas con el tamaño de los objetos de código en los sistemas más modernos no tiene sentido. La alta densidad de código era una buena opción de optimización para un sistema multiusuario con una memoria principal cara, como Multics.

### 2.1.2 Utilerías y herramientas del sistema

Las utilerías de UNIX son todos aquellos comandos, lenguajes o programas que ayudarán a la labor de los usuarios y administradores de sistemas.

El sistema operativo Unix, desde su concepción, tiene la filosofía de la especialización, es decir, está compuesto por muchos pequeños programas, cada uno de los cuales realiza una función muy específica. De hecho, son pocos los comandos de Unix que realizan más de una función en particular. Unix provee los medios necesarios para

integrar distintos comandos, aprovechando sus capacidades conjuntas para llevar a cabo tareas más complejas de una manera relativamente sencilla.

Como ejemplo de utilerías tenemos:

nn: Produce una tabla de símbolos a partir de un archivo objeto (.o o .out).
size: Nos da el número de bytes usados por el código ejecutable, los datos y la pila.
ar: Construye librerías .a para /lib o /usr/lib.
adb: Controla la ejecución de un proceso y permite examinar el archivo core (imagen de memoria y registros del CPU) equivale al debug de MS-DOS.
m4: Programa preprocesador de propósito general (intercambia valores constantes por sus identificadores dentro del archivo fuente). Expansión de macros.
make: Emplea una estructura de archivos de comandos como una ayuda a la construcción del programa. El archivo de programas se llamará makefile. Está constituido por llamadas a compilación que dependen de la modificación de los archivos fuentes.
top: Proporciona una vista dinámica en tiempo real de los procesos ejecutándose en el sistema.
sed: editor de línea de texto.
sort: Da orden a la salida estándar.
grep: Despliega las líneas que se ajustan al parámetro solicitado.
egrep: Despliega las líneas que se ajustan al o los parámetros solicitados.
awk: Lenguaje de programación que puede combinarse ampliamente con la salida de comandos o utilerías.

### 2.1.3 Permisos y privilegios del sistema

En Unix la seguridad de los diferentes objetos del sistema esta determinada por el usuario y el grupo al que este pertenecen dichos objetos. Internamente, estas entidades el sistema las representa a través de dos números, el uid - user id, para el usuario - y el gid - group id, para el grupo -, que representa a un conjunto de usuarios.

Cada usuario pertenece a un grupo primario y puede pertenecer a varios grupos secundarios.

En los sistemas de archivos Unix tradicionales, esta información se almacena en los archivos /etc/passwd y /etc/group.

En los Unix más recientes, no se accede a ellos directamente, sino que existe toda una serie de módulos - conocidos como PAM - que permiten obtener la información necesaria de otras fuentes como pueden ser NIS, LDAP u otro sistema de almacenamiento.

Los permisos que pueden existir sobre los objetos de los sistemas de archivos son:

- Lectura (r,Read). Permiso de lectura.
- Escritura (w,Write). Permiso de escritura.
- Ejecucion (x,eXecute/search). Permiso de ejecución. Cuando se aplica a directorios, sirve para que pueda buscarse un archivo en los mismos.
- Setuid (setuid). Permite que el objeto al cual se le establezca este permiso se ejecute con el perfil de dueño del archivo, no con el perfil del usuario que lo esta ejecutando.
- Setuid (setgid). Igual que setuid, pero para los grupos.

- Sticky bit (t). Herencia de los antiguos sistemas Unix. Aplicado a un archivo, hacía que los procesos que arrancaran así, no se fueran al swap. Aplicado a un directorio, hace que los archivos que existan dentro de un directorio, puedan borrarlos sólo el root o el propietario - frente al caso normal, donde cualquiera con permisos de escritura pudiera hacerlo.

En conclusión en un entorno Unix cada archivo tiene un conjunto de permisos asociados con él, los que determinan qué puede hacerse con el archivo y quién puede hacerlo.

Los permisos de cada archivo son la protección más básica que se le puede asignar a este tipo de objetos del sistema operativo, derivado de esto, la importancia y cuidado con la que dichos permisos deben ser asignados. La mala configuración o definición de un permiso puede representar un hueco de seguridad el cual puede ser aprovechado por un atacante para obtener privilegios de forma no autorizada.

Unix controla los privilegios de usuarios o cuentas en base al UID no en base al nombre de cuenta. El usuario con mayores privilegios dentro del sistema se designa con el valor 0 de UID y generalmente se le nombra como root, pero esto únicamente de forma convencional, debido a que los privilegios están directamente relacionado sólo al UID.

De lo anterior se deriva la importancia de no otorgar mayores permisos a los archivos que los que estrictamente se requiera para su correcto uso y operación. Un ejemplo de permisos excesivos sería dar permisos de escritura a un archivo que le perteneciera a root y el cual se ejecutará de cuando en cuando. Un atacante interno podría modificar el archivo para que se ejecutase código maligno de manera arbitraria.

### 2.1.4 Lenguaje C

C es un lenguaje de programación de propósito general que es ubicado como lenguaje de bajo y medio nivel. Sus principales características son:

- Programación estructurada.
- Economía de las expresiones.
- Abundancia en operadores aritméticos y tipos de datos.
- Codificación en alto y bajo nivel simultáneamente.
- Reemplaza ventajosamente la programación en ensamblador (assembler).
- Utilización natural de las funciones primitivas del sistema.
- No está orientado a ningún área en especial (lenguaje de propósito general).
- Producción de código objeto altamente optimizado.
- Manejo de coma flotante, registros, interrupciones, vectores, uniones y más.
- Independiente de la plataforma.

El lenguaje de programación C fue creado por Dennis Ritchie a principios de los 70's como un lenguaje implementado para el naciente sistema operativo UNIX, del cual casi todos sus programas y comandos fueron reescritos en C.

C se derivó de los lenguajes de programación BCPL de Martín Richards y B de Ken Thompson y evolucionó de ser un pequeño lenguaje estructurado desarrollado sobre una diminuta máquina a ser uno de los lenguajes dominantes hasta el día de hoy.

Una de las peculiaridades de C es su riqueza de operadores, puede decirse que prácticamente dispone de un operador para cada una de las posibles operaciones en código máquina.

A pesar de que C, fue pensado para ser altamente portable y para programar lo inprogramable, posee al igual que otros lenguajes algunos inconvenientes:

- Carece de instrucciones de entrada/salida, de instrucciones para manejo de cadenas de caracteres, con lo que este trabajo se deja para la biblioteca de rutinas, con la consiguiente pérdida de portabilidad.
- La excesiva libertad en la escritura de los programas puede llevar a errores en la programación que, por ser correctos sintácticamente no se detectan a simple vista.
- Por otra parte las precedencias de los operadores convierten a veces las expresiones en pequeños rompecabezas.
- El programador debe llevar el control de los recursos utilizados.
- Aunque C es altamente portable y compatible con muchas plataformas, su código debe ser compilado para generar los archivos objeto y después los ejecutables. En gran número de casos, algunas variables de entorno y tipos de datos deben ser ajustados para lograr la compilación.
- C hace de un tarea sencilla algo complicado y largo (dado que obliga al programador a pensar en los detalles, en lugar de solo concentrarse en algoritmo de solución), auque el resultado final probablemente sea mucho más eficiente, el tiempo necesario para programar una tarea en C es en la mayoría de los casos, mayor que en cualquier otro lenguaje.

A pesar de todo, C ha demostrado ser un lenguaje extremadamente eficaz y expresivo.

C++ a pesar de abarcar tres importantes paradigmas, la programación estructurada, la programación genérica y la programación orientada a objetos, adolece de muchos de los mismo problemas que C, es decir, todavía gran cantidad de las tareas necesitan ser programadas, ésto a pesar de contar con un gran número de bibliotecas, además de que las definiciones de tipos de datos entre plataformas continua siendo un problema.

### 2.1.5 Lenguaje Perl

Perl, Lenguaje práctico para la extracción de datos y generación de informes, es un lenguaje de programación diseñado por Larry Wall creado en 1987. Perl toma características del C, del lenguaje interpretado shell (sh), AWK, sed, Lisp y, en un grado inferior, muchos otros lenguajes de programación.

Estructuralmente, Perl está basado en un estilo de bloques como los del C o AWK, y fue ampliamente adoptado por su destreza en el procesado de texto y no tener ninguna de las limitaciones de los otros lenguajes de script.

Larry Wall comenzó a trabajar en Perl en 1987 mientras trabajaba como programador en Unisys y anunció la versión 1.0 en el grupo de noticias comp.sources.misc el 18 de diciembre de 1987. El lenguaje se expandió rápidamente en los siguientes años. Perl 2, liberado en 1988, aportó un mejor motor de expresiones regulares. Perl 3, liberado en 1989, añadió soporte para datos binarios.

Hasta 1991 la única documentación de Perl era una simple (y cada vez más larga) página de manual Unix. En 1991 se publicó Programming Perl o Programación en Perl (el libro del dromedario de O'Reilly) y se convirtió en la referencia de facto del lenguaje. Al mismo tiempo, el número de versión de Perl saltó a 4, no por marcar un gran cambio en el lenguaje, sino por identificar a la versión que estaba documentada en el libro.

Perl 5 fue liberado el 17 de octubre de 1994. Fue casi una completa reescritura del intérprete y añadió muchas nuevas características al lenguaje, incluyendo objetos, referencias, paquetes y módulos. Los módulos proveen de un mecanismo para extender el lenguaje sin modificar el intérprete. Esto permitió estabilizar su núcleo principal, además de permitir a los programadores de Perl añadirle nuevas características.

El 26 de octubre de 1995, se creó el Comprehensive Perl Archive Network (CPAN). CPAN es una colección de sitios web que almacenan y distribuyen fuentes en Perl, binarios, documentación, scripts y módulos. Originalmente, cada sitio CPAN debía ser accedido a través de su propio URL; hoy en día, http://www.cpan.org redirecciona automáticamente a uno de los cientos de repositorios espejo de CPAN.

En 2007, Perl 5 continua siendo mantenido. Características importantes y algunas construcciones esenciales han sido añadidas, incluyendo soporte Unicode, Hilos (threads), un soporte importante para la programación orientada a objetos y otras mejoras.

Perl se llamó originalmente "Pearl", por la Parábola de la Perla. Larry Wall quería darle al lenguaje un nombre corto con connotaciones positivas; asegura que miró (y rechazó) todas las combinaciones de tres y cuatro letras del diccionario. También consideró nombrarlo como su esposa Gloria. Wall descubrió antes del lanzamiento oficial que ya existía un lenguaje de programación llamado PEARL y cambió la ortografía del nombre.

El nombre normalmente comienza con mayúsculas "Perl" cuando se refiere al lenguaje y con minúsculas "perl" cuando se refiere al propio programa intérprete.

El nombre es descrito ocasionalmente como "PERL" (por Practical Extraction and Report Language). Aúnque esta expansión ha prevalecido en muchos manuales actuales, incluyendo la página del manual de Perl, Perl es un retroacrónimo y oficialmente el nombre no quiere decir nada.

Descripción

Perl es un lenguaje de propósito general originalmente desarrollado para la manipulación de texto y que ahora es utilizado para un amplio rango de tareas incluyendo administración de sistemas, desarrollo web, programación en red, desarrollo de GUI y más.

Perl compila primero de manera rápida el programa completo en un formato intermedio. Como cualquier otro compilador, el compilador de Perl realiza varias optimizaciones y proporciona retroalimentación instantánea acerca de errores sintácticos, semánticas o de ligado de bibliotecas. Una vez que el compilador de Perl aprueba la sintaxis, semántica y bibliotecas necesarias, pasa el código intermedio al intérprete para su ejecución. Aun que todo esto tal vez pueda sonar un poco más complicado y largo que el procedimiento de un script de shell, el compilador e intérprete de Perl son muy eficientes.

Por último gracias a los manejadores de archivos y a los arreglos de Perl, es posible extraer y dar formato a los datos dentro de ciclos altamente eficientes.

Características

La estructura completa de Perl deriva ampliamente del lenguaje C. Perl es un lenguaje imperativo, con variables, expresiones, asignaciones, bloques de código delimitados por llaves, estructuras de control y subrutinas.

Perl también toma características de la programación shell. Todas las variables son marcadas con un signo precedente (sigil). Los sigil identifican inequívocamente los nombres de las variables, permitiendo a Perl tener una rica sintaxis. Notablemente, los sigil permiten interpolar variables directamente dentro de las cadenas de caracteres (strings). Como en los shell, Perl tiene muchas funciones integradas para tareas comunes y para acceder a los recursos del sistema.

Perl toma las listas del Lisp, hash (memoria asociativa) del AWK y expresiones regulares de sed. Todo esto simplifica y facilita todas las formas del análisis sintáctico, manejo de texto y tareas de gestión de datos.

En Perl 5, se añadieron características para soportar estructuras de datos complejas, funciones de primer orden y un modelo de programación orientada a objetos. Ésto incluyen referencias, paquetes y una ejecución de métodos basada en clases y la introducción de variables de ámbito léxico, que hizo más fácil escribir código robusto. Una característica principal introducida en Perl 5 fue la habilidad de empaquetar código reutilizable como módulos. Larry Wall indicó más adelante que "la intención del sistema de módulos de Perl 5 era apoyar el crecimiento de la cultura Perl en vez del núcleo de Perl".

Todas las versiones de Perl hacen el tipificado automático de datos y la gestión de memoria. El intérprete conoce el tipo y requerimientos de almacenamiento de cada objeto en el programa; reserva y libera espacio para ellos según sea necesario. Las conversiones legales de tipo se hacen de forma automática en tiempo de ejecución; las conversiones ilegales son consideradas errores fatales.

Diseño

El diseño de Perl puede ser entendido como una respuesta a tres amplias tendencias de la industria informática: rebaja de los costes en el hardware, aumento de los costes laborales y las mejoras en la tecnología de compiladores. Anteriormente, muchos lenguajes de ordenador como el Fortran y C, fueron diseñados para hacer un uso eficiente de un hardware caro. En contraste, Perl es diseñado para hacer un uso eficiente de los costosos programadores de computación.

Perl tiene muchas características que facilitan la tarea del programador a costa de requerimientos de CPU y memoria mayores, éstos incluyen gestión de memoria automática, tipificado dinámico, strings, listas y hashes, expresiones regulares, introspección y la función eval().

Larry Wall fue adiestrado como un lingüista y el diseño de Perl ha sido muy aleccionado con principios lingüísticos. Ejemplos incluyen la Codificación Huffman (las construcciones más comunes deben ser las más cortas), buena distribución (la información importante debe ir primero) y una larga colección de primitivas del lenguaje. Perl favorece las construcciones del lenguaje, tan naturales, como para los humanos son la lectura y la escritura, incluso si eso hace más complicado al intérprete Perl.

La sintaxis de Perl refleja la idea de que "cosas que son diferentes deben parecer diferentes". Por ejemplo, escalares, arreglos y hashes tienen diferente sigil. Índices de arreglos y claves hash usan diferentes clases de paréntesis. Strings y expresiones regulares tienen diferentes delimitadores estándar.

Perl tiene características que soportan una variedad de paradigmas de programación, como la procedural, funcional y la orientada a objetos. Al mismo tiempo, Perl no obliga a seguir ningún paradigma en particular, ni obliga al programador a elegir alguna de ellas.

Hay un amplio sentido de lo práctico, tanto en el lenguaje Perl como en la comunidad y la cultura que lo rodean. El prefacio de Programming Perl comienza con, "Perl es un lenguaje para tener tu trabajo terminado". Una consecuencia de esto es que Perl no es un lenguaje ordenado, tolera excepciones a las reglas y emplea la heurística para resolver ambigüedades sintácticas. Debido a la naturaleza indulgente del compilador, a veces los errores pueden ser difíciles de encontrar.

Perl tiene varios lemas que transmiten aspectos de su diseño y uso, uno de ellos es: There's more than one way to do it (Hay más de una forma de hacerlo). Una meta de Perl es hacer las cosas fáciles de forma fácil y las tareas difíciles, posibles.

Aplicaciones

Perl tiene muchas y variadas aplicaciones, gracias a la disponibilidad de muchos módulos estándares y de terceras partes.

Se ha usado desde los primeros días del Web para escribir guiones (scripts) CGI. Es una de las "tres Pes" (Perl, Python y PHP), que son los lenguajes más populares para la creación de aplicaciones Web, y es un componente integral de la popular solución LAMP para el desarrollo web. Grandes proyectos escritos en Perl son Slash, IMDb[2] y UseModWiki, un motor de Wiki. Muchos sitios web con alto tráfico, como Amazon.com y Ticketmaster.com usan Perl extensamente.

Perl se usa a menudo como un "lenguaje pegamento", ligando sistemas e interfaces que no fueron diseñados específicamente para interoperar; y para "la extracción de datos", convirtiendo o procesando grandes cantidades de datos de manera fácil y rápida. Perl es una popular herramienta de propósito general para los administradores de sistemas, especialmente en programas pequeños que pueden ser escritos y ejecutados en una sola línea de comandos.

Implementación

Perl está implementado como un intérprete, escrito en C, y posee un gran conjunto de módulos, escritos en Perl y C. La distribución fuente tiene, en el 2005, 12 MB cuando se empaqueta y comprime en un fichero tar. El intérprete tiene 150.000 líneas de código C y se compila en un ejecutable de 1 MB en las arquitecturas de hardware más típicas. De forma alternativa, el intérprete puede ser compilado como una biblioteca y ser embebida en otros programas. Hay cerca de 500 módulos en la distribución, sumando 200.000 líneas de Perl y unas 350.000 líneas adicionales de código C. Mucho del código C en los módulos consiste en tablas de codificación de caracteres.

Perl es un lenguaje dinámico y tiene una gramática sensible al contexto que puede quedar afectada por el código ejecutado durante una fase de ejecución intermedia. Por eso Perl no puede ser analizado sintáctica y lexicográficamente con un sencilla combinación de analizadores léxicos/sintáctico Lex/Yacc. En cambio, el intérprete

implementa su propio analizador léxico, que coordinado con un analizador sintáctico modificado de GNU, bison resuelve las ambigüedades del lenguaje. Se ha dicho que "sólo perl puede parsear o analizar a Perl". La razón de esto se atestigua por las persistentes imperfecciones de otros programas que emprenden la tarea de parsear Perl, como los analizadores de código y los auto-indentadores, que tienen que vérselas no sólo con las muchas formas de expresar inequívocamente construcciones sintácticas, sino también con el hecho de que también Perl no puede, generalmente, ser parseado sin antes ser ejecutado.

Disponibilidad

Perl es software libre y está licenciado bajo la Licencia Artística y la GNU General Public License. Existen distribuciones disponibles para la mayoría de sistemas operativos. Está especialmente extendido en Unix y en sistemas similares, pero ha sido portado a las plataformas más modernas (y otras más obsoletas). Con sólo seis excepciones confirmadas, puede ser compilado en todos los Unix, compatibles con POSIX ó cualquier otra plataforma Unix compatible. Sin embargo, esto no es normalmente necesario, porque Perl está incluido por defecto en la instalación la mayoría de los sistemas operativos Unix y Linux.

Estructura del lenguaje

Programa ejemplo

En Perl, el programa canónico "Hola mundo" es:

```
#!/usr/bin/perl -w
use strict;
print "¡Hola mundo!\n";    # "\n" es un 'nueva línea'
```

La primera línea contiene el shebang (“#!”, par de caracteres que identifica el texto que sigue), que le indica al sistema operativo dónde encontrar el intérprete de Perl. La segunda línea introduce el pragma strict que se usa en grandes proyectos de software para el control de calidad. La tercera imprime el string ¡Hola mundo! y un carácter de nueva línea. Sigue un comentario ( "\n" es un 'nueva línea' ).

El signo # en la tercera línea es un 'token comentario', que permite al intérprete perl ignorar todo lo que le siga, hasta el final de la línea de código.

El shebang es la forma normal para invocar al intérprete en los sistemas Unix. Los sistemas Windows pueden seguir utilizándolo o pueden asociar la extensión de archivo .pl con el intérprete Perl. Algunos editores de texto también usan la línea shebang como una pista sobre el modo de trabajo en que deben operar. Si el programa es ejecutado por perl y no invocado por el shell, la línea que empieza por el shebang es parseada para interpretar las opciones. En otro caso, es ignorada.

Tipos de datos

Perl tiene tres tipos de datos: escalares, listas y hashes:

- Un escalar es un sólo valor; puede ser un número, un string (cadena de caracteres) o una referencia.
- Una lista es una colección ordenada de escalares (una variable que almacena una lista se llama array).

- Un hash, o memoria asociativa, es un mapeo de strings a escalares; los strings se llaman claves y los escalares valores.

Todas las variables están precedidas por un sigil, que identifica el tipo de dato que es accedido (no el tipo de dato de la misma variable). Se puede usar el mismo nombre para variables de diferentes tipos, sin que tengan conflictos.

### 2.1.6 Lenguaje PHP

Historia

PHP fue originalmente diseñado y desarrollado en base a un conjunto de scripts de Perl por el programador danés-canadiense Rasmus Lerdorf en el año 1994 para mostrar su currículum vitae y guardar ciertos datos, como la cantidad de tráfico que su página web recibía. Derivado del incremento de funcionalidad requerido Rasmus escribió implementaciones de C cada vez más largas, así como un grupo de CGIs binarios escritos también en el lenguaje C, además de agregar capacidades de comunicación con bases de datos, lo que permitió a los usuarios desarrollar aplicaciones dinámicas de web de manera simple. El 8 de junio de 1995 fue publicado como “Herramientas de Página Personal” o "Personal Home Page Tools" después de que Lerdorf lo combinara con su propio Intérprete de Forma o Form Interpreter para crear PHP/FI.

PHP/FI incluyo algunas de las funcionalidades básicas de lo que hoy conocemos como PHP. Tenía variables tipo Perl, interpretación automática de variables de forma y sintaxis HTML embebida. La sintaxis como tal era similar a la de Perl no obstante mucho más limitada, simple y algo inconsistente.

En noviembre de1997 PHP/FI versión 2.0 fue oficialmente liberada. Diseñada y escrita también en lenguaje C. Para ese entonces contaba ya con un culto de algunos miles de usuarios alrededor del mundo, con aproximadamente 50,000 dominios que reportaban tenerlo instalado. Lo que representaba cerca del 1% del total de dominios en la Internet en ese momento. Aunque había grupos de personas contribuyendo con pedazos de código al proyecto, éste era en su mayoría todavía el trabajo o proyecto de un sólo hombre

Rasmus decidió liberar el código fuente de PHP/FI para que cualquiera pudiera usarlo, corregirlo y mejorarlo.

Dos programadores israelíes del Technion, Zeev Suraski y Andi Gutmans, reescribieron el analizador sintáctico (*parser* en inglés) en el año 1997 y crearon la base del PHP versión 3.0. Esto después de analizar el PHP/FI y encontrarlo carente y poco poderoso para el desarrollo de una aplicación eCommerce en la que ellos estaban trabajando para un proyecto de la universidad. PHP versión 3.0 fue la primera versión que realmente semejaba ser algo parecido al PHP que conocemos ahora,

El nuevo PHP fue liberado bajo un significado completamente diferente de nombre; el nuevo significado del nombre removía toda implicación o referencia al uso personal limitado que el significado de PHP/FI versión 2.0 suponía. Fue nombrado simplemente PHP, con el significado de un acrónimo recursivo:

PHP Hypertext Preprocessor (Procesador de Hipertexto PHP).

Inmediatamente comenzaron experimentaciones públicas de PHP versión 3.0 y fue publicado oficialmente en junio del 1998.

Para 1999, Suraski y Gutmans reescribieron el código de PHP, produciendo lo que hoy se conoce como Zend Engine o motor Zend, una combinación de los nombres de ambos, Zeev y Andi. También fundaron Zend Technologies en Ramat Gan, Israel.

Una de las fortalezas de PHP 3.0 fue su gran extensibilidad de características, aunado a la capacidad de proporcionar a los usuarios finales una sólida infraestructura para gran número de diferentes bases de datos, protocolos y APIs.

Las características extendidas de PHP 3.0 atrajeron a docenas de desarrolladores los cuales contribuyeron con nuevos módulos. Derivado de esto PHP 3.0 logró un tremendo éxito. El soporte de sintaxis orientada a objetos y la gran consistencia del lenguaje, fue también otro de los logros de PHP 3.0.

A finales de 1998, PHP creció hasta ser un número de decenas de miles de usuarios y miles de sitios web reportando trabajar con él. La estadística indica que en su pico, llego a tener el 10% de cobertura en los servidores de web de la Internet.

PHP 4

En invierno de 1998, poco después de la liberación oficial de PHP 3.0, Andi Gutmans y Zeey Suraski iniciaron la reescritura del motor de PHP. Los objetivos de diseño eran mejorar el desempeño de aplicaciones complejas y el mejoramiento de la modularidad del código base. PHP versión 3.0 trajo consigo el soporte para una amplia variedad de aplicaciones bases de datos, pero no estaba diseñado para manejar estas complicadas aplicaciones de manera eficiente.

El nuevo motor denominado Zend, fue por primera vez introducido a mediados de 1999. PHP 4.0 se basó en este motor y en un amplio repertorio de características adicionales. Fue oficialmente liberado en mayo de 2000, casi dos años después de su predecesor. Así mismo PHP 4.0 incluyó soporte para muchos más servidores de HTTP, sesiones, buffer de salida, mejores y más seguras formas de manejar los datos de entrada de los usuarios y nuevas construcciones del lenguaje.

Hoy en día PHP es utilizado por cientos de miles de desarrolladores, por millones de sitios y se estima que más del 20% de los dominios de Internet trabajan con PHP.

PHP 5 fue liberado en Julio de 2004 después de un largo desarrollo y múltiples pre-liberaciones.

Características de PHP

- Rapidez de ejecución.
- Es un lenguaje específicamente diseñado para realizar aplicaciones Web, mientras que otros lenguajes son adaptaciones de lenguajes preexistentes, no pensados para la Web.
- El software necesario para ejecutar aplicaciones es software libre. (GNU/Linux)
- Mantiene un bajo consumo de recursos de máquina.
- Gran seguridad, muy poca probabilidad de corromper los datos.
- Trabaja con gran diversidad de bases de datos.
- Rico en funciones predefinidas.

- Puede ser instalado en servidores Windows (Con emuladores apache+php+(MySql/Postgres).
- Fácil aprendizaje.
- Es un lenguaje libre.
- Trabaja en combinación con otras tecnologías: perl, javascript, phyton y dhmtl
- Puedes hacerlo todo, por ejemplo, creación de gráficos interactivos por el usuario, al estilo photoshop, creación de pdfs.
- Creación o uso de binarios ejecutables por parte del propio usuario.
- Permite embeber sus pequeños fragmentos de código dentro de la página HTML.
- Las tareas fundamentales que puede realizar directamente el lenguaje son definidas en el mismo lenguaje como funciones.
- Presenta una filosofía totalmente diferente y, con un espíritu más generoso, es progresivamente construido por colaboradores desinteresados que implementan nuevas funciones en nuevas versiones del lenguaje.
- Combina excelentemente con otras herramientas, como son el servidor apache y la base de datos mysql y posgres, todas ellas gratuitas.
- Buena documentación.

### 2.1.7 Intérpretes de línea de comandos

Intérprete de línea de comandos Bourne-Shell (sh)

*Bourne Shell* era el intérprete de línea de comandos por defecto de la versión 7 de Unix, y sustituyó al intérprete de comandos Thompson o Thompson shell, cuyo ejecutable tenía el mismo nombre: sh. Fue desarrollado por Stephen Bourne, de los Laboratorios Bell de AT&T, y vio la luz en la versión 7 de Unix distribuida a colegios y universidades. A pesar de ser un intérprete de comandos con algunas carencias, es todavía utilizado en los sistemas operativos tipo Unix de algunos vendedores. En la mayoría de los sistemas Unix el programa o binario de la Bourne shell o un programa compatible se encuentra en /bin/sh.

Los principales objetivos de Bourne Shell eran aprovechar dos características claves del kernel de la versión 7:

- La lista de parámetros (argumentos) mucho más larga, limitada a 8192 bytes (anteriormente 127).
- Las variables de entorno. Éstas eran una nueva característica de la versión 7 y permitía pasar mucha información a los programas a través del arranque.

El intérprete de línea de comandos Bourne obtuvo importantes logros, fue el primero en destacar la convención de usar el descriptor de archivo 2 para mensajes de error, permitiendo un control del programa mucho mayor durante la creación de un script o guión de ejecución, manteniendo los mensajes de error separados de la información.

Otras innovaciones importantes de sh fueron:

- La sustitución de comandos utilizando los acentos invertidos `comando`.
- *Here documents*, utilizando el operador << para ingresar un bloque de texto dentro de un script.
- Iteración a través de "*for ~ do ~ done*", en particular el uso de $* para iterar utilizando los argumentos pasados al script.

- El mecanismo de selección "`case ~ in ~ esac`", principalmente utilizado para el parser de argumentos.
- *`sh`* proporcionó soporte para las variables de ambiente y exportación de variables

Aunque pretendía ser un intérprete de comandos interactivo, ganó popularidad como un lenguaje de scripting con la publicación por parte de Brian W. Kernighan y Rob Pike de "*The UNIX Programming Environment"*. Éste fue el primer libro publicado comercialmente que presentaba al intérprete de comandos como un lenguaje de programación en forma de tutorial.

A través de los años el intérprete de comandos Bourne fue impulsado por AT&T y sus variables de Unix (Version7, SystemIII, SVR2, SVR3, SVR4). Debido a que el intérprete de comandos Bourne nunca fue versionado, la única manera de identificarlo es probando sus características.

Stephen Bourne introdujo en sh algunos aspectos del compilador ALGOL 68C con el que él había estado trabajando en la Universidad de Cambridge. Reutilizó notablemente porciones de las estructuras y cláusulas "**`if`** `~ then ~ elif ~ else ~` **`fi`**", "**`case`** `~ in ~ out ~` **`esac`**" y "`for ~ while ~` **`do`** `~` **`od`**" de ALGOL 68 en la sintaxis de sh.

Sh se encuentra presente de en casi todos los Unix, pero en algunos casos ya no es el intérprete de por defecto, y en los Linux en algunos casos es tan sólo una liga que apunta al intérprete de comandos bash.

Intérpretes de línea de comandos C shell (Csh) y TC shell (Tcsh)

El Csh es un intérprete de comandos desarrollado por Hill Joy para el sistema operativo Unix BSB. Fue originalmente basado en la sexta edición de intérprete de comandos del Unix BSD y fue presentado como predecesor del intérprete de comando Bourne Shell. Su sintaxis esta modelada en base al lenguaje de programación C. El intérprete de comandos C o Csh agrega muchas mejoras de características con respecto a su predecesor el intérprete de comandos Bourne; tales como alias e historial de comandos. Hoy en día el intérprete de comandos C original no es utilizado en los sistemas Unix tan ampliamente como en el pasado y ha sido sobrepasado y sustituido por otros intérpretes de comandos como el Tenex C shell (tcsh) basado originalmente en el código del intérprete de comandos Csh pero agregando el autocompletado de nombres de archivos y la edición de línea de comandos.

Tcsh conocido también como Tshell es un intérprete de comandos basado y compatible con el Csh. Es comparable a los intérpretes de comandos Ksh o bash (Korn Shell y GNU Bourne-Again-Shell). Desarrollado y actualizado de manera independiente al Csh.

La 't' de tcsh proviene de la T en TENEX, un sistema operativo que inspiró a Ken Greer, el autor de tcsh, a implementar la característica de auto-completar. Tcsh fue desarrollado a principios de los 70's en la Universidad de Mellon y este trabajo fue continuado por Paul Placeway de la Universidad del Estado de Ohio.

Otras características comunes a ambos intérpretes de comandos, son la habilidad para suspender y reiniciar de manera interactiva trabajos en cualquier momento, así como de enviar éstos a background, variables indexadas numéricamente, la expansión

y búsqueda de directorios hogar por medio del operador '~' (el cual fue tan popular que fue también asimilado por los servidores de Web) y la inclusión de operadores aritméticos los cuales no existían en sh excepto a través de llamadas a aplicaciones externas.

Muchos proveedores de Unix y Mac OS anteriormente proporcionaban el tcsh como interprete de comando por defecto. Pero poco a poco este ha sido remplazado por bash.

Csh y Tcsh son intérpretes de comandos que hoy en día están cada vez más en desuso a pesar de sus numerosas características, esto debido principalmente y sobre todo en el caso de Tcsh a cuestiones de mercado.

Interprete de línea de comandos Korn o Korn Shell (Ksh)

El shell o intérprete de línea de comandos Korn es un intérprete de comandos de Unix, el cual fue desarrollado por David Korn del los laboratorios AT&T a principios de los 80's. También conocido como Ksh es muy similar y compatible con el intérprete de línea comandos Bourne y también incluye muchas características del Cshell como el historial de comandos el cual fue inspirado a raíz de las requisiciones de los usuarios de los laboratorios Bell.

Una de las principales ventajas de ksh sobre los intérpretes de comandos tradicionales de Unix, es en su uso como lenguaje de programación. Desde su concepción muchas características le fueron agregadas gradualmente, siempre manteniendo gran compatibilidad con el interprete de comandos Bourne (también identificado como sh).

Ksh se apega al estándar de lenguaje shell (POSIX 1003.2 "Comité de lenguaje de Interprete de comandos y Utilerías").

Hasta el año 2000 el intérprete de comandos Korn era un software propietario de AT&T. A partir de esa fecha se convirtió en software tipo "open source". Debido principalmente a que Korn era una marca registrada y sólo se comercializaba junto con los Unix de la arquitectura AT&T un número de interprete de comandos de tipo "open source" surgieron como alternativas. Entre estas podemos nombrar al pdksh, al zsh y por supuesto a Bourne-Again-Shell mejor conocido como bash.

Aun y cuando la versión de Korn Shell, la Ksh93 trajo consigo múltiples mejoras (arreglos asociativos, aritmética de punto flotante), algunos vendedores (Solaris y hp) continuaron ofreciendo la versión anterior el ksh88.

Desgraciadamente aunque Korn Shell agrega cuestiones de punto flotante y arreglos asociativos (conceptos similares a los utilizados actualmente por Perl), esta ultima característica no siempre esta disponible y depende de la versión instalada.

Más aún el Korn Shell presenta un gran problema en el ámbito iterativo, que generalmente sólo es visible cuando el ciclo es lo suficientemente grande o largo. Este problema deriva de la manera en que los valores dentro de los ciclos de control se cargan y se despachan. El procesamiento o acción que se realiza en el bloque de código es muy lento, debido a que las variables dentro de los ciclos de control recogen todos los valores asignados antes iniciar a despacharlos, por lo que se utiliza el doble tiempo necesario que el que se utilizaría por ejemplo para el mismo ciclo en el lenguaje C.

Por ejemplo, para el ciclo siguiente, el tiempo total para imprimir los valores que la variable “$var” toma en cada uno de los ciclos de impresión, no es muy diferente que el tiempo utilizado por el lenguaje C en un programa con código similar al presentado aquí.

```
for var in one two three ;
  do
    echo $var
  done
```

Esto es debido principalmente a que el número de valores que toma la variable “$var” es muy pequeño y la diferencia de tiempos entre el programa ejecutable en C y el script en Ksh sería de 1 o 2 milésimas de segundo a favor del programa ejecutable en C. Esto como ya he mencionado es debido a la manera en que ksh recolecta los datos que serán introducidos al bloque de ciclo de control por la variable, en este caso “$var”.

En el caso del script de Korn, los la variable “$var” se carga con los datos “one two three” antes de iniciar el ciclo. Dichos valores serán pasados al bloque de ciclo de control uno a uno hasta agotar la variable “$var”.

Por el contrario el caso de un ejecutable de C en el que se halla programado el mismo ciclo, los valores se cargaran y serán pasados al ciclo de control uno a uno.

En la actualidad el intérprete de comandos Ksh continua siendo utilizado, pero en muchos de los sistemas Unix no es el intérprete de comandos por defecto.

Intérprete de línea de comandos Bourne-Again-Shell (Bash)

Bash es un intérprete de comandos de Unix escrito para el proyecto GNU. Su nombre es una acrónimo de *Bourne-Again-Shell* (otro shell bourne) haciendo un juego de palabras con born-again que significa nacer otra vez o renacer en referencia al intérprete de comandos bourne (sh), el cual como hemos visto es uno de los primeros shells o intérpretes de comandos importantes de Unix. El Bourne shell o intérprete de comandos Bourne se desarrollo hacia 1978 por Stephen Bourne (por entonces investigador de los Laboratorios Bell) y fue distribuido con la versión 7 de Unix.

El intérprete de comandos bash fue escrito por Brian Fox en 1987. En 1990, Chet Ramey se convirtió en su principal desarrollador. Bash es el intérprete de comandos por defecto en la mayoría de los sistemas GNU/Linux, además de los Mac OS X Tiger, y puede ejecutarse en la mayoría de los sistemas tipo Unix.

La sintaxis de órdenes ó comandos de bash es un superconjunto de la sintaxis del intérprete de comandos Bourne (sh).

La mayoría de los shell scripts (guiones de órdenes) Bourne pueden ejecutarse por bash sin ningún cambio, con la excepción de aquellos scripts de shell Bourne que hacen referencia a variables especiales de Bourne o que utilizan una orden interna de Bourne. La sintaxis de órdenes de bash incluye ideas tomadas desde el Korn Shell (ksh) y el C Shell (csh), como la edición de la línea de órdenes, el historial de órdenes, la pila de directorios, las variables $RANDOM y $PPID, y la sintaxis de substitución de órdenes POSYX: $(...). Cuando se utiliza como un intérprete de órdenes interactivo, bash proporciona autocompletado de nombres de programas, nombres de archivos, nombres de variables, etc, cuando el usuario pulsa la tecla TAB.

La sintaxis de bash tiene muchas extensiones que no proporciona el shell Bourne (sh). Varias de las mencionadas extensiones se enumeran a continuación.

Matemáticas con enteros

Una gran limitación del intérprete de comandos Bash es que no puede realizar cálculos con enteros sin lanzar un proceso externo. En cambio, un proceso bash puede realizar cálculos con enteros utilizando la orden ((...)) y la sintaxis de variables $[...] de la siguiente manera:

```
VAR=55              # Asigna el valor entero 55 a la variable VAR.
((VAR = VAR + 1))   # Suma uno a la variable VAR.  Observe la ausencia
                    # del carácter '$'.
((++VAR))           # Otra forma de sumar uno a VAR.  Preincremento
                    # estilo C.
((VAR++))           # Otra forma de sumar uno a VAR.  Postincremento
estilo C.
echo $[VAR * 22]    # Multiplica VAR por 22 y substituye con resultado.
echo $((VAR * 22)) # Otra forma de realizar lo mismo.
```

La orden ((...)) también se puede utilizar en sentencias condicionales, ya que su código de retorno es 0 o 1 dependiendo de si la condición es cierta o falsa:

```
if ((VAR == Y * 3 + X * 2)) then
        echo Si
fi
((Z > 23)) && echo Si
```

La orden ((...)) soporta los siguientes operadores relacionales: '==', '!=', '>', '<', '>=', y '<='.

Uno de los grandes inconvenientes de bash, es que no puede realizar cálculos en coma flotante. Los únicos intérpretes de comandos de Unix capaces de esto son el Korn Shell (versión de 1993) y el zsh (a partir de la versión 4.0) y el Tcsh.

Las expresiones regulares son una parte importante de los lenguajes de programación, y como tal bash también soporta un sin fin de expresiones regulares que le permiten ajustar sus filtros y tareas a parámetros no definidos.

Bash es sin duda por muchas de las características, uno de los lenguajes de programación elegido para realizar la programación del motor de monitoreo, además de que éste puede ser encontrado en casi todos sistemas operativos de tipo Unix, más sin embargo, también tiene el mismo problema en el ámbito de los ciclos. Para solucionar este problema es necesario recurrir a otro lenguaje de programación como Perl.

### 2.1.8 Ejecución de comandos de forma remota

En su inicio los equipos en las red eran sólo unos pocos y la relación de administradores y servidores era casi univoca, es decir, casi uno a uno, por lo que no existía la necesidad de mecanismo de administración remota. Conforme el tamaño de las redes y el número de computadoras en ellas aumento, surgió la necesidad de desarrollar herramientas que hicieran posible la ejecución de comandos de manera remota, con el propósito de realizar las tareas de administración sin la necesidad de

desplazarse hasta el servidor, el cual podía estar a unos cuando metros o a kilómetros de distancia.

Telnet

Telnet fue una de las primeras implementaciones de este tipo de mecanismos. Telnet es el n ombre del protocolo y del programa que lo implementa, el cual fue desarrollado en 1969. Es un protocolo estándar de propósito general y bidireccional que proporciona servicio de comunicaciones orientado a bytes de 8 bits. Permite comunicar terminales y procesos orientados a Terminal, de manera predeterminada el demonio de telnet atiende peticiones el en puerto 23.

Telnet permite la comunicación múltiple tanto individual como conexiones de múltiples usuarios que tenga cuenta en la máquina.

Telnet es una herramienta útil, pero tiene dos carencias que hacen que su uso hoy día sea considerado altamente riesgoso.

- No cifra ninguno de los datos enviados sobre la conexión, incluyendo los nombres de las cuentas y las contraseñas, por lo que estos pueden ser interceptados por un tercero para ser utilizados más adelante con propósitos maliciosos.
- Carece de un esquema de autentificación, lo cual no permite garantizar que la comunicación se este llevando acabo entre las dos partes deseadas (emisor-receptor) y no entre las partes deseadas y un intermediario.

rlogin y rsh

Los programas rlogin y rlogind proporcionan servicios de terminal remota similares a los ofrecidos por telnet; rlogin es el programa cliente y rlogind es el programa servidor, este último atiende las peticiones en el puerto 513.

Existen dos diferencias importantes entre rlogin y telnet:

- rlogind no requiere que el usuario teclee su nombre de cuenta; la cuenta de usuario se transmite automáticamente al inicio de la conexión.
- Si la conexión proviene de una “máquina confiable” o de un “usuario confiable”, la máquina a la que se esta conectando, permite que el usuario ingrese al sistema sin la necesidad de teclear su contraseña.

El programa rlogin permite a un usuario conectarse a un equipo a través de la red y ejecutar comandos en el equipo remoto (únicamente Unix), es orientado a procesos de terminal y fue distribuido por primera vez en la versión de Unix de Berkeley (4.2BSD). En general rlogin se utilizaba en entornos de redes corporativas o académicas, donde el nombre de la cuenta de usuario se compartía entre todas las máquinas Unix en la red (a menudo utilizando NIS).

A este tipo de esquemas, se les denomina ambientes o entornos de confianza y para el caso de rlogin básicamente sólo se necesita que el equipo remoto aparezca en el archivo /etc/hosts.equiv, o que el usuario en cuestión tenga un archivo .rhosts en el “directorio hogar” o “home directory” del equipo remoto.

rlogin tiene algunos serios problemas de seguridad:

- Toda la información incluyendo los nombres de cuentas y las contraseñas son transmitidas en texto plano (haciéndolo vulnerable a la intercepción de los datos).
- El archivo .rlogin o .rhosts puede fácilmente ser mal configurado (permitiendo que potencialmente cualquiera pueda ingresar a la máquina sin la necesidad de una contraseña), por esta razón muchos administradores prohíben el uso de archivos .rlogin.
- El protocolo parcialmente se basa en la información que honestamente proporciona el cliente de rlogin (incluyendo puerto y nombre de la máquina de origen). Un cliente corrupto fácilmente puede forzar estos datos a valores arbitrarios para ganar acceso, dado que el protocolo de rlogin no tiene medios para autenticar la identidad de otras máquinas o asegurar que el cliente de rlogin en una máquina sea realmente el cliente de rlogin que dice ser.
- Aunque una mejora de rlogin fue confiar la autenticación del nombre de las máquinas al servicio de DNS. Este tampoco resulto ser más seguro, dado que el DNS no fue diseñado para ser un protocolo seguro, y él mismo no tiene manera de garantizar que la información que proporciona es correcta o no.
- Otra mejora fallida de rlogin fue implementar el nslookup reverso y posteriormente un doble nslookup reverso lo cual hizo más complejo el spoofing para los atacantes pero no implosible.
- La práctica común de montar los directorios hogar de los usuarios vía NFS expone a rlogin a ataques a través de archivos .rhosts falsos. Lo cual también significa que cualquier falla de seguridad de NFS automáticamente afecta a rlogin.

Debido a este serio problema rlogin fue poco usado a través de redes no seguras como Internet. Derivado de estos problemas, la mayoría de los Unix modernos ya no lo incluyen de manera predeterminada en sus paquetes de instalación.

El paquete original de Berkeley rlogin estaba acompañado de rcp (copia remota por las palabras en inglés *remote-copy*) y de rsh (shell remoto por las palabras en inglés remote shell, el cual permitía la ejecución de comandos en la máquina remota sin la necesidad de que el usuario se registrara en el sistema), los cuales compartían el esquema de control de acceso basado en los archivos hosts.equiv y .rhosts pero conectándose a un demonio diferente el rshd. Estos programas sufrían de los mismos problemas de seguridad.

El programa ssh (shell seguro por sus palabras en inglés *secure shell*) representa un adecuado sustituto del rcp y del rsh con el scp y el ssh respectivamente, debido a que este implementa comunicación cifrada para la privacidad de los datos y autenticación por medio de par de llaves.

En conclusión, la ejecución de comandos de manera remota es una tarea indispensable para los administradores, por lo cual se acude a mecanismos que implementen las tareas requeridas, pero las necesidades cambian constantemente y una de estas necesidades es la seguridad, por lo cual mecanismos útiles en el pasado, resultan obsoletos hoy en día.

### 2.1.9 Shells restringidos y comunicación cifrada

Para hablar de shell restringido, por principio es necesario definir que es un shell y hablar un poco de la estructura de Unix.

La estructura de Unix habla a grandes rasgos de un sistema operativo esquematizado en capas concéntricas, en la que el hardware esta en el centro y le siguen en orden el Kernel o núcleo, las llamadas al sistema junto con los shell's y los programas de usuario.

El Kernel es la parte central de sistema operativo, él controla y administra los recursos de la computadora. El Kernel da seguimiento de los programas que están siendo ejecutados y es el encargado de inicializar a cada usuario en el sistema. Sin embargo el Kernel no interactúa con los usuarios para interpretar sus órdenes o comandos. Es el programa denominado Shell (también conocido como intérprete de comandos o intérprete de línea de comandos), el programa que el Kernel utiliza para ejecutar las órdenes de los usuarios.

Se le denomina shell debido a que oculta de tras de su interfase los detalles internos del sistema operativo, en contraste con el Kernel el cual hace referencia a componentes de bajo nivel o más internos al sistema operativo.

Cuando uno se registra en el sistema, es el Kernel el encargado de revisar si el nombre de la cuenta y la contraseña son correctos y después ejecuta el shell asignado al usuario para que éste interactué a través de él con el Kernel.

Los shells restringidos son utilizados para minimizar el daño que una cuenta en el sistema pueda ocasionar.

Un shell restringido es una shell de Unix que ha sido modificado para restringir el entorno, los comandos a ejecutar por un usuario de Unix. Puede configurarse para permitirle al usuario ejecutar sólo ciertos programas y para impedirle cambiar de directorio.

En Unix, esta modalidad es invocada a través de la opción –r del shell o con el comando rsh (shell restringido, en inglés *restricted shell*). Cuando rsh inicia, ejecuta los comandos en el archivo $HOME/.profile. Una vez que el archivo .profile es procesado, las siguientes restricciones toman efecto:

- El usuario no puede cambiar de directorio.
- El usuario no puede cambiar el valor de la variable de ambiente $PATH.
- El usuario no puede utilizar comandos que contengan /.

Los shells restringidos pobremente implementados pueden ser rotos o vulnerados, obteniendo acceso a un entorno más allá del entorno definido por el shell restringido. Generalmente esto se logra recurriendo a características de algún programa a través de las cuales puedan ser burladas las restricciones. Un buen ejemplo de esto es el comando vi.

El shell restringido del usuario puede iniciar el vi y después utilizar este comando de la siguiente manera:

:set shell=/bin/sh *y después utilizar el comando* :shell

Lo cual permitiría que ejecutase comandos en el entorno del shell /bin/sh y con ello haber logrado romper su entorno restringido.

Si el shell restringido permite al usuario definir un editor, por ejemplo ejecutando "/bin/csh –i –f". Entonces el usuario podría al igual que en el caso anterior tener acceso a mayores privilegios que los otorgados.

Por ejemplo en el caso de que al usuario no se le permita leer archivos, el usuario puede intentar abrir el archivo adentro del programa de correo.

Otro ejemplo podría ser cuando al usuario no se le dan permisos para editar archivos, en este caso el usuario podría intentar salvar los archivos a través del programa de correo.

También si al usuario se le impide el uso del comando cd para evitar que cambie directorio, el usuario puede intentar utilizar el FTP en su cuenta y así cambiar a otros directorios.

El FTP significa también otro riesgo para los shells restringidos, dado que también pueden ser utilizados para editar archivos. Esto se logra trayendo el archivo que se desea editar desde una locación externa (desde la que previamente se edito libremente el archivo), a la ruta en la que el archivo original se encuentra, lo cual sobrescribiría al archivo original con el archivo editado y nuevamente el entorno y seguridad del shell restringido seria sobrepasada.

El intérprete de comandos bash también tiene una modalidad de shell restringido.

Si el interprete de comandos Bash es invocado con el nombre rbash o si la opción *"–restricted"* se agrega a la invocación normal de bash, el shell se convierte en restringido.

Al igual que el los casos anteriores un shell restringido se utiliza para establecer entornos más controlados que los proporcionados por los shells estándar.

Las características adicionales que el shell restringido de bash proporciona son:

- No se puede establecer o limpiar los valores establecidos de las variables SHELL, PATH, ENV o de las variables BASH_ENV.
- No se puede especificar el nombre de un archivo que contenga "/" como parámetro de algún comando.
- No se pude importar definiciones de funciones desde variables de ambiente de inicio.
- No se puede utilizar la redirección utilizando los operadores de redirección >, >|, <>, >&, &> y >>.
- No se pueden agregar o eliminar comandos con las opciones *–f* o *–d* del comando enable.
- Apagar el modo restringido con el modo 'set +r' o 'set +o restricted'.

El shell restringido es un buen esquema para limitar las acciones de la mayoría de los usuarios y circunscribirlos a un entorno, sin embargo su implementación debe se bien planeada y estar acorde a las tareas y sólo a las tareas que se desea que el usuario realice y debe ser extensamente evaluado para verificar que este tipo de vulnerabilidades no estén presentes.

## 2.2 Protocolos

En redes informáticas, un protocolo es el lenguaje o conjunto de reglas formales que especifican el intercambio de datos entre dispositivos o computadoras (nodos). El protocolo es una convención, estándar, o acuerdo entre partes, que regula la conexión, la comunicación y la transferencia de datos entre dos sistemas. En su forma más simple, un protocolo se puede definir como las reglas que gobiernan la semántica (significado de lo que se comunica), la sintaxis (forma en que se expresa) y la sincronización (quién y cuándo transmite) de la comunicación.

Los protocolos pueden estar implementados bien en hardware (tarjetas de red), software (drivers), o una combinación de ambos.

Al hablar de protocolos no se puede generalizar, debido a la gran amplitud de campos que cubren, tanto en propósito, como en especificidad. No obstante, la mayoría de los protocolos especifican una o más de las siguientes propiedades:

- Detección de la conexión física sobre la que se realiza la conexión (cableada o sin cables).
- Pasos necesarios para comenzar a comunicarse (Handshaking).
- Negociación de las características de la conexión.
- Cómo se inicia y cómo termina un mensaje.
- Formato de los mensajes.
- Qué hacer con los mensajes erróneos o corruptos (corrección de errores).
- Cómo detectar la pérdida inesperada de la conexión, y qué hacer en ese caso.
- Terminación de la sesión de conexión.
- Estrategias para asegurar la seguridad (autenticación, cifrado).

### 2.2.1 Protocolos TCP/IP

Historia

El Protocolo de Internet (IP) y el Protocolo de Transmisión (TCP), fueron desarrollados inicialmente en 1973 por el informático estadounidense Vinton Cerf como parte de un proyecto dirigido por el ingeniero norteamericano Robert Kahn y patrocinado por la Agencia de Programas Avanzados de Investigación (*ARPA*, siglas en inglés) del Departamento Estadounidense de Defensa. Internet comenzó siendo una red informática de ARPA (llamada ARPAnet) que conectaba redes de computadoras de varias universidades y laboratorios en investigación en Estados Unidos.

Introducción

El mal llamado protocolo TCP/IP es un realidad una familia de protocolos de red. Este conjunto de protocolos, llega a ser más de 100, entre los cuales podemos encontrar protocolos como el HTTP, el FTP, el ARP, el POP, el TELNET, el SMTP, el ICMP y claro esta el protocolo IP y el protocolo TCP de los cuales toma su nombre el conjunto de protocolos.

Arquitectura y Modelo TCP/IP

El modelo de OSI de referencia consiste de siete capas que representan una división funcional de las tareas requeridas para implementar una red, el modelo OSI es una herramienta conceptual que se utiliza para analizar como varias tecnologías y

protocolos se ajustan para la implementación de redes. Sin embargo no es el único modelo de red que intenta dividir tareas en capas y componentes.

La suite de TCP/IP fue de hecho creada antes que el modelo de referencia OSI,y por ende sus creadores no utilizaron el modelo de referencia OSI para explicar la arquitectura TCP/IP.

Modelo TCP/IP

Los desarrolladores de la suite de protocolos TCP/IP crearon su propio modelo de arquitectura para ayudar a describir sus componentes y funciones. Este modelo también conocido como modelo DARPA en referencia a la agencia encargada de su desarrollo.

Más haya del modelo que se utilice para representar las funciones de una red o de como se le llame, las funciones que le modelo representa son muy similares en su naturaleza. En TCP/IP y OSI existe una correspondencia, aunque esta no es siempre uno a uno.

El modelo TCP/IP utiliza cuatro capas las cuales empalman con su equivalente en las seis capas superiores del modelo OSI de referencia. El modelo TCP/IP no cubre la capa física, debido a que la capa de enlace de datos esta considerada como el punto en que la interfase entre la pila de TCP/IP y el hardware de red ocurre.

La figura 2.1 muestra la relación que guardan las capas del modelo TCP/IP con el modelo de referencia OSI.

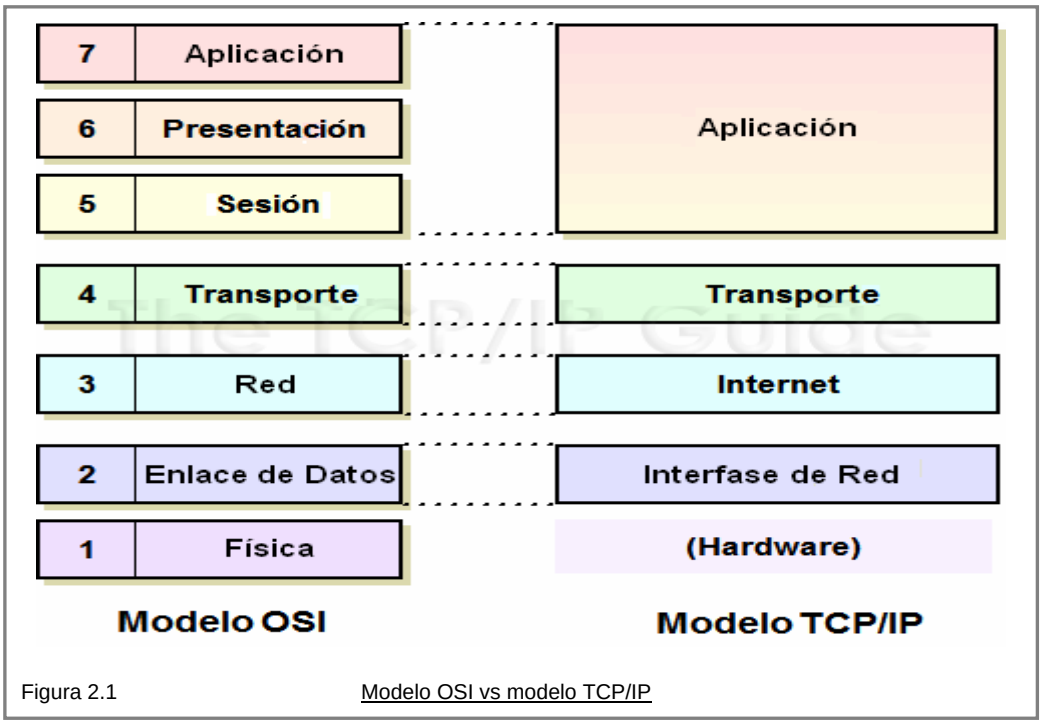


Figura 2.1 Modelo OSI vs modelo TCP/IP

La siguiente tabla muestra los niveles del modelo TCP/IP una breve descripción y los protocolos asociados al nivel correspondiente.

| Nivel | Descripción | Protocolos Involucrados |
|---|---|---|
| Aplicación | Define los protocolos de aplicación TCP/IP y cómo se conectan los programas de host a los servicios del nivel de transporte para utilizar la red | HTTP, Telnet, FTP, TFTP, SNMP, DNS, SMTP, X |
| Transporte | Permite administrar las sesiones de comunicación entre equipos. Define el nivel de servicio y el estado de la conexión utilizada al transportar datos. | TCP, UDP, RTP |
| Internet | Empaqueta los datos en datagramas IP, que contienen información de las direcciones de origen y destino utilizada para reenviar los datagramas entre hosts y a través de redes. Realiza el enrutamiento de los datagramas IP. | IP, ICMP, ARP, RARP |
| Interfaz de red | Especifica información detallada de cómo se envían físicamente los datos a través de la red, que incluye cómo se realiza la señalización eléctrica de los bits mediante los dispositivos de hardware que conectan directamente con un medio de red, como un cable coaxial, un cable de fibra óptica o un cable de cobre de par trenzado. | Ethernet, Token Ring, FDDI, X.25, Frame Relay, RS-232, v.35 |

Tabla descripción de capas del modelo TCP/IP

## Protocolo IP

El Protocolo de Internet IP (por sus siglas en inglés *Internet Protocol*), es un protocolo no orientado a conexión usado tanto por el origen como por el destino para la comunicación de datos a través de una red de paquetes conmutados.

Los datos en una red basada en IP son enviados en bloques conocidos como paquetes o datagramas, aún que este último enunciado no es realmente cierto dado que la definición de un datagrama es un fragmento de un paquete que es enviado con la suficiente información como para que la red pueda simplemente encaminar el fragmento hacia el Equipo Terminal de Datos receptor o DTE receptor (por sus iniciales en inglés *Data Terminal Equipment*).
En particular, en IP no se necesita ninguna configuración antes de que une equipo intente enviar paquetes a otro con el que no se había comunicado antes.

El protocolo IP no proporciona mecanismos de comunicación confiable. No hay reconocimientos ni fin-a-fin ni salto-a-salto. No existe un control de errores de datos (únicamente a una sumatoria de comprobación de la cabecera). No exite la retransmisión ni un flujo de control.

Dicho de otro forma, el protocolo de Internet no es del todo fiable y por ello se le conoce también como best effort (mejor esfuerzo), el protocolo de Internet no provee ningún mecanismo para determinar si un paquete o datagrama alcanza o no su destino y únicamente proporciona (como ya se menciono) sumas de comprobación de sus cabeceras y no de los datos transmitidos, por lo que no es posible garantizar nada sobre la recepción del paquete, éste podría llegar dañado, en otro orden con respecto a otros paquetes, duplicado o simplemente no llegar. Para hablar de fiabilidad, es necesario recurrir a otros protocolos de la capa de transporte, como el TCP.

Los errores detectados también pueden ser reportados vía el protocolo de control de mensajes de Internet ICMP el cual esta implementado en el módulo del protocolo IP.

Si la información a transmitir ("datagramas") supera el tamaño máximo "negociado" (MTU) en el tramo de red por el que va a circular, el paquete original podría ser dividido en paquetes más pequeños (datagramas), y reensamblado luego cuando sea

necesario. Estos fragmentos podrán ir cada uno por un camino diferente dependiendo de como estén de congestionadas las rutas en cada momento.

La función o propósito del Protocolo de Internet es mover paquetes o datagramas a través de un conjunto de redes interconectadas. Esto se logra mediante el envío de paquetes o datagramas de un módulo de Internet a otro hasta que el destino es alcanzado. Los módulos de Internet residen en los hosts y en los gateways en el sistema de Internet. Los paquetes o datagramas son encaminados o ruteados de un módulo de Internet a otro a través de redes individuales basándose en la interpretación de la dirección de Internet, de lo que se deriva la importancia de las direcciones de Internet dentro del protocolo de Internet.

Las direcciones de Internet se componen de cuatro octetos de 32 bits. Una dirección inicia con el número de la red seguido de la dirección local llamada o denominada el resto. Tres de las principales clases de direcciones de Internet son:

Clase A, en la cual el bit de mayor orden es cero y los 7 bits restantes son la red y finalmente los restantes 24 bits son la dirección local.

Clase B, en la cual la cual los dos bits de mayor orden o más significativos son uno-cero, los siguientes 14 bits son la red y los últimos 16 bits son la dirección local.

Clase C, en la cual los tres bits de mayor orden o más significativos son uno-uno-cero, los 21 bits siguientes son la red y los últimos 8 bits son la dirección local.

Las cabeceras IP contienen las direcciones de las máquinas de origen y destino (direcciones IP), direcciones que serán usadas a su vez por los conmutadores de paquetes (switches) y los ruteadores (routers) para decidir el tramo de red por el que reenviarán los paquetes o datagramas.

El IP es el elemento común en la Internet de hoy. El actual y más popular protocolo de red es IPv4. IPv6 es el sucesor de IPv4.
Hoy en día el número de dispositivos que requieren de una dirección IP crece aún ritmo muy acelerado; previendo el agotamiento de las direcciones IP disponibles, desde hace ya algún tiempo se desarrollo y el IPv6, el cual utiliza direcciones IP de fuente y destino de 128 bits lo cual agrega un rango de direcciones mucho mas amplio.

Historia TCP

El TCP tuvo su nacimiento en los inicios de los setentas y surgió debido a las carencias de su predecesor, el Protocolo de Control de Red o NCP (por sus siglas en ingles, Network Control Protocol), el cual operaba en la red DARPA o ARPAnet y realizaba tareas similares a las que hoy en día realizan el TCP y el IP.

Debido a las limitantes en el NCP, se inicio el desarrollo de un nuevo protocolo que se ajustarse mejor al crecimiento las nuevas redes interconectadas. Este nuevo protocolo se formalizo primero en el RFC 675 y fue llamado Programa de Control de Transmisión de Internet (TCP). Al igual que su antecesor, el TCP era responsable básicamente de todo lo necesario para permitir que las aplicaciones se ejecutaran en la red. Así, por principio TCP fue ambas, TCP e IP.

Después de varios años de ajustes y revisiones del TCP, en el año de 1977 surgió la versión 2, y con ella surgieron problemas referentes al concepto y estructura inicial del protocolo.

El TCP manejaba los datagramas de transmisión y de enrutamiento (funciones de capa tres), así como las conexiones, la confiabilidad y la administración del flujo de datos (funciones de capa cuatro) lo cual significaba que el TCP violaba los principios clave de modularidad y capas del protocolo. TCP forzaba a todas las aplicaciones a utilizar las funciones de capa cuatro para utilizar las funciones de la capa tres. Esto hacia que TCP fuera inflexible y pobremente ajustable para las necesidades de las aplicaciones que sólo requerían funciones de nivel bajo y no de las funciones de alto nivel.

Como resultado, se tomo la decisión de separar el TCP en dos.

Las funciones de capa cuatro se mantuvieron en el renombrado Protocolo de Control de Transmisión TCP al igual que su antecesor, pero esta vez por las siglas Transmition Control Protocol o Protocolo de Control de Transmisión.

Las funciones de la capa tres se convirtieron en el Protocolo de Internet. Esta división se finalizó en la versión 4 de TCP y por ende la primera versión de IP fue denominada IPv4 en consistencia con la versión 4 del TCP.

La versión 4 de TCP fue definida en el RFC 793 y publicada en septiembre de1981.

TCP incluye un extenso conjunto de mecanismos para asegurar que los datos lleguen del origen al destino de manera confiable, consistente y rápida. Una de las claves para esta operación, es el sistema de reconocimiento de ventana deslizable, el cual permite a cada dispositivo llevar un seguimiento de cuales bytes de datos han sido enviados y confirmar al emisor que los datos que han sido recibidos. Los datos sin reconocimiento son eventualmente retransmitidos de manera automática. Este mismo sistema proporciona características de buffer y flujo de control.

Cinco de las funciones más importantes de TCP son:

- Direccionamiento y multiplexado.
- Establecimiento, Manejo y Terminación de Conexiones.
- Manejo de datos y empaquetado.
- Confiabilidad y Servicios de Transmisión de Calidad.
- Flujo de Control y características que evitan la congestión.

### 2.2.2 SNMP versión 1

Originalmente en Internet, la administración de redes se hacía usando el SGMP (por sus iniciales en inglés *Simple Gateway Monitoring Protocol*). Luego, se definió el Protocolo Simple de Administración de Red o SNMP por sus siglas en inglés (*Simple Netword Management Protoco*l) para la administración de redes y dispositivos de red.

El SNMP es un protocolo de la capa de aplicación el cual es encapsulado en el protocolo UDP y facilita el intercambio de información de administración entre dispositivos de red. Es parte de la suite de protocolos TCP/IP. SNMP permite a los administradores supervisar el desempeño de la red, buscar y resolver sus problemas, y planear su crecimiento.

SNMP necesita, debajo de él, el soporte de las capas de transporte (proveyendo multiplexación y demultiplexación de servicios, y checksum para confiabilidad) y de red (ruteo entre redes, protección de entidades SNMP de las diferencias del medio físico, fragmentación y reensamblado de paquetes).

Una red administrada por SNMP consiste de tres principales componentes:

- Dispositivos administrados.
- Agentes.
- Sistemas de Administración de Red o NMS por sus siglas en inglés (*Network Management Systems*).

Los objetos a ser administrados se definen en la Base de Información de Administración (MIB por sus siglas en ingles, Management Information Base). Estos objetos deben seguir un determinado conjunto de reglas como las mencionadas en la Estructura de Información de Administración (SMI por sus siglas en ingles, Structure Management Information).

Un dispositivo administrado es un nodo de red que contiene un agente SNMP y reside en una red administrada. Los dispositivos administrados son también aquellos elementos que brindan información sobre su estado, como puede ser por ejemplo una central de alarma de incendio, un equipo de transmisión por fibra óptica, etc. Los dispositivos administrados, recolectan y almacenan información y ponen esta información a disposición de los NMS's utilizando el SNMP. Los dispositivos administrados, algunas veces llamados elementos de red, pueden se ruteadores, computadoras o impresoras etcétera.

Un agente es un módulo de software de administración de red que reside en un dispositivo administrado. Un agente posee un conocimiento local de información de administración (memoria libre, número de paquetes IP recibidos, rutas, etcétera), la cual es traducida a un formato compatible con SNMP y organizada en jerarquías
Un NMS ejecuta aplicaciones que supervisan y controlan a los dispositivos administrados. Los NMS’s proporcionan el volumen de recursos de procesamiento y memoria requeridos para la administración de la red. Uno o más NMS’s deben existir en cualquier red administrada.

Comandos básicos de SNMP

Los dispositivos administrados son supervisados y controlados usando cuatro comandos SNMP básicos: lectura, escritura, notificación y operaciones transversales.

El comando de lectura es usado por un NMS para supervisar elementos de red. El NMS examina diferentes variables que son mantenidas por los dispositivos administrados.

El comando de escritura es usado por un NMS para controlar elementos de red. El NMS cambia los valores de las variables almacenadas dentro de los dispositivos administrados.

El comando de notificación es usado por los dispositivos administrados para reportar eventos en forma asíncrona a un NMS. Cuando cierto tipo de evento ocurre, un dispositivo administrado envía una notificación al NMS.

Las operaciones transversales son usadas por el NMS para determinar que variables soporta un dispositivo administrado y para recoger secuencialmente información en tablas de variables, como por ejemplo, una tabla de rutas.

Base de información de administración SNMP (MIB)

Una Base de Información de Administración (MIB) es una colección de información que está organizada jerárquicamente. Las MIB's son accedidas usando un protocolo de administración de red, como por ejemplo, SNMP.

Un objeto administrado (algunas veces llamado objeto MIB) es uno de cualquier número de características específicas de un dispositivo administrado. Los objetos administrados están compuestos de una o más instancias de objeto, que son esencialmente variables.

Existen dos tipos de objetos administrados: Escalares y tabulares. Los objetos escalares definen una simple instancia de objeto. Los objetos tabulares definen múltiples instancias de objeto relacionadas que están agrupadas conjuntamente en tablas MIB.

Un ejemplo de un objeto administrado es atInput, que es un objeto escalar que contiene una simple instancia de objeto, el valor entero que indica el número total de paquetes AppleTalk de entrada sobre una interfaz de un ruteador.

Un identificador de objeto (object ID) únicamente identifica un objeto administrado en la jerarquía MIB. La jerarquía MIB puede ser representada como un árbol con una raíz anónima y los niveles, que son asignados por diferentes organizaciones. La figura 2.2 muestra este concepto.

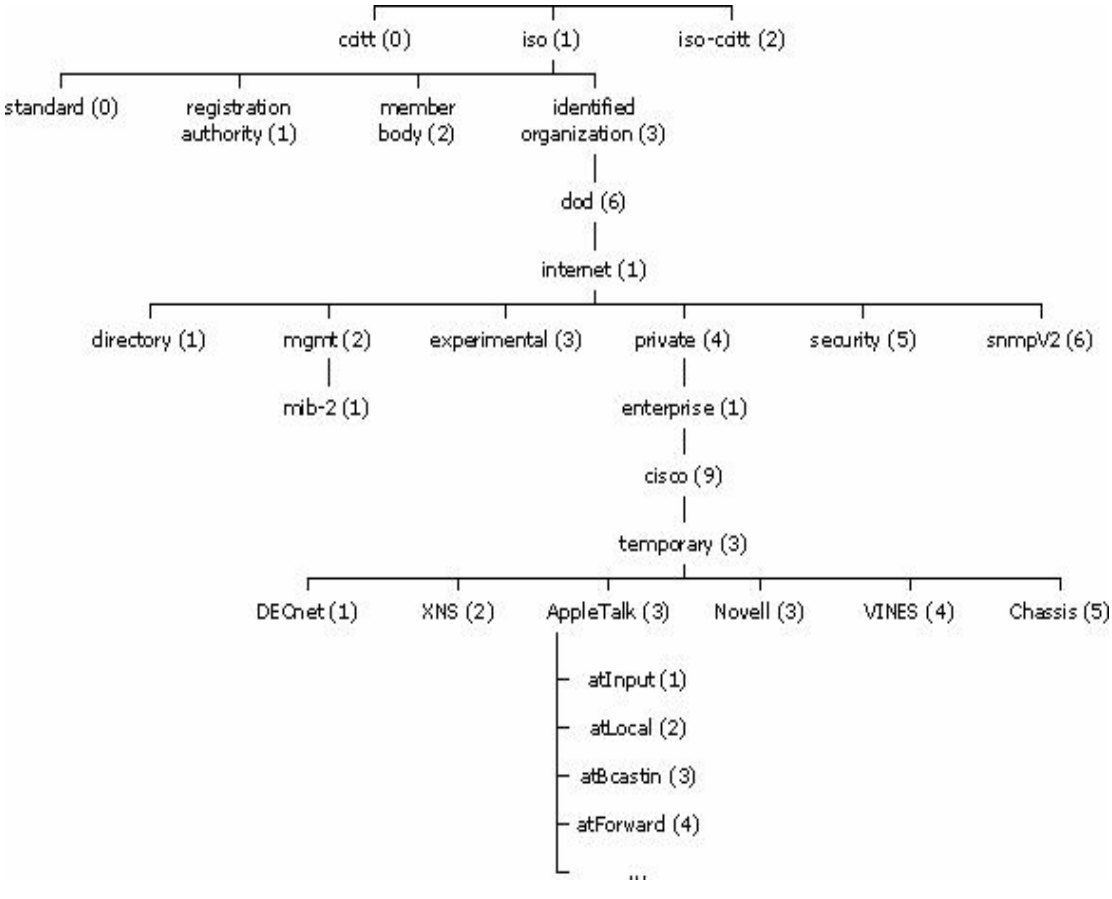


Figura 2.2 Árbol jerárquico MIB

El árbol MIB ilustra las variadas jerarquías asignadas por las diferentes organizaciones

Los identificadores de los objetos ubicados en la parte superior del árbol pertenecen a diferentes organizaciones estándares, mientras los identificadores de los objetos ubicados en la parte inferior del árbol son colocados por las organizaciones asociadas.

Los vendedores pueden definir ramas privadas que incluyen los objetos administrados para sus propios productos. Las MIB's que no han sido estandarizadas típicamente están localizadas en la rama experimental.

El objeto administrado atInput podría ser identificado por el nombre de objeto iso.identified-organization.dod.internet.private.enterprise.cisco.temporary.AppleTalk.atInput o por el descriptor de objeto equivalente 1.3.6.1.4.1.9.3.3.1.

El corazón del árbol MIB se encuentra compuesto de varios grupos de objetos, los cuales en su conjunto son llamados mib-2. Los grupos son los siguientes:

- System (1).
- Interfaces (2).
- AT (3).
- IP (4).
- ICMP (5).
- TCP (6).
- UDP (7).
- EGP (8).
- Transmission (10).
- SNMP (11).

Es importante destacar que la estructura de una MIB se describe mediante el estándar Notación Sintáctica Abstracta 1 (Abstract Syntax Notation One).

Mensajes SNMP

Para realizar las operaciones básicas de administración anteriormente nombradas, el protocolo SNMP utiliza un servicio no orientado a la conexión (UDP) para enviar un pequeño grupo de mensajes (UDPs) entre los administradores y agentes. La utilización de un mecanismo de este tipo asegura que las tareas de administración de red no afectarán al rendimiento global de la misma, ya que se evita la utilización de mecanismos de control y recuperación como los de un servicio orientado a la conexión, por ejemplo TCP.

Los puertos comúnmente utilizados para SNMP son los mostrados en la siguiente tabla:

| No. Puerto | Descripción |
|---|---|
| 161 | SNMP |
| 162 | SNMP-trap |

Tabla de puertos por defecto de SNMP

Seguridad y SNMP

A pesar de sus virtudes el protocolo SNMP está considerado dentro de la lista de las 20 vulnerabilidades más importantes en Internet. Según la versión del protocolo SNMP utilizada, los mecanismos de autenticación son extraordinariamente simples. Esto, unido a la posibilidad de modificar la configuración de los dispositivos de la red, lo convierte en un importante agujero en la seguridad corporativa.

### 2.2.3 SNMP versión 2

SNMP v2 añade algunas nuevas posibilidades a la versión anterior de SNMP, de las cuales, la más útil para los servidores es la operación get-bulk. Ésta permite que se envíen un gran número de entradas MIB en un sólo mensaje, en vez de requerir múltiples consultas get-next para SNMP v1. En principio la versión 2 de SNMP se presento con mejoras para la seguridad pero éstas presentaron huecos de seguridad aún mayores que los anteriores y una nueva especificación de SNMP v2 fue liberada en 1996 sin ninguna facilidad de seguridad.

### 2.2.4 SNMP v3

La versión 3 de SNMP incorpora capacidades de configuración remota así como seguridad. La arquitectura SNMPv3 introduce el modelo de seguridad basado en usuario USM (por sus iniciales en inglés *User-based Security Model*) y el modelo de vista basada en el control de acceso VACM (por sus iniciales en inglés *View-based Access Control Model*), para el control de acceso. La arquitectura soporta diferentes controles de acceso y modelos de procesamiento de mensajes tales como:

- Seguridad.
- Autenticación.
- Autorización y control de acceso.
- Marco Administrativo.
- Nombramiento de entidades.
- Personal y políticas.
- Nombres de usuarios y claves de administración.
- Notificación de destinos.
    - Relaciones con proxies.
- Configurable de forma remota vía operaciones de SNMP.

SNMPv3 también introduce la capacidad de configurar dinámicamente al agente de SNMP utilizando el conjunto de comandos de SNMP en el MIB de los objetos que representan la configuración del agente. La configuración dinámica permite las operaciones de inserción, modificación y eliminación de elementos de la configuración tanto local como remota.

## 2.3 Comunicación y seguridad

Día a día cada vez más servicios de diferente índole son puesto a disposición de los usuarios vía red, esto trae consigo un sin fin de beneficios tanto para ellos como para los organizaciones o instituciones que los proporcionan, sin embargo estos servicios utilizan protocolos de comunicación los cuales no fueron diseñados en principio para proporcionar seguridad (TCP, UDP, IP). Lo mismo sucede con los protocolos

desarrollados para el monitoreo de red y monitoreo de dispositivos (SNMP) debido a esto, el monitoreo en algunos casos puede resultar más perjudicial que benéfico dado que la información que de los dispositivos o servidores se extrae viaja en claro y puede ser interceptada por terceros no autorizados los cuales pueden utilizar esta valiosa información para planificar mejores ataques, más aún, en la mayoría de los casos el robo de esta información no es detectado, dándole al atacante un rango de acción mucho más amplio.

La necesidad de transmitir información de manera segura a llevado al desarrollo de protocolos que permitan implementar tanto cifrado como autenticación en la comunicación, uno de estos protocolos es ssh el cual se basa en ssl para proporcionar seguridad.

### 2.3.1 Ataques y riesgos de seguridad

Las amenazas o ataques a la seguridad, están directamente relacionados a los objetivos de seguridad antes descritos.

La severidad de los ataques no depende del objetivo seguridad con el que este relacionado (integridad, disponibilidad, confidencialidad o autenticidad).

La severidad de un ataque de seguridad depende de muchos factores: El valor que el dueño o dueños de la información den a la misma. El valor que el atacante de a la información obtenida o el valor y repercusiones de la difusión de dicha información sin autorización explicita. La pérdida y repercusiones inmediatas y/o posteriores que la interrupción de un servicio genere. La pérdida y repercusiones inmediatas y/o posteriores que la modificación y/o perdida de la información genere.

Un acto de esta naturaleza puede tener consecuencias incuantificables.

### 2.3.2 Tipos de algoritmos de cifrado

Encriptado o cifrado son dos palabras que utilizan para indicar que un texto en claro será modificado de forma tal que será ilegible, excepto para aquellos que puedan revertir el cifrado y obtener el texto en claro nuevamente.

Cifra es una antigua palabra árabe para designar el número cero; en la antigüedad cuando Europa empezaba a cambiar del sistema de numeración romano al arábigo, se desconocía el cero por lo que este resultaba misterioso y de ahí que cifrado signifique misterioso.

La criptografía (del griego kryptos, «ocultar», y graphos, «escribir», literalmente «escritura oculta») es el arte o ciencia de cifrar y descifrar información utilizando técnicas matemáticas que hagan posible el intercambio de mensajes de manera que sólo puedan ser leídos por las personas a quienes van dirigidos.

Propiamente la ciencia de la criptología engloba las técnicas de cifrado o criptografía así como el criptoanálisis, que se encarga de romper los textos cifrados en ausencia de la clave.

Con frecuencia los procesos de cifrado y descifrado se encuentran en la literatura como encriptado y desencriptado, aúnque ambos son neologismos-anglicismos de los términos en inglés *encrypt y decrypt*- todavía sin reconocimiento académico.

La historia de la criptografía es larga y abunda en anécdotas. Ya las primeras civilizaciones desarrollaron técnicas para enviar mensajes durante las campañas militares de forma que si el mensajero era interceptado la información que portaba no corriera el peligro de caer en manos del enemigo. Posiblemente, el primer criptosistema que se conoce fuera documentado por el historiador griego Polibio: un sistema de sustitución basado en la posición de las letras en una tabla.

También los romanos utilizaron sistemas de sustitución, siendo el método actualmente conocido como César, porque supuestamente Julio César lo utilizó en sus campañas, uno de los más conocidos en la literatura (según algunos autores, en realidad Julio César no utilizaba este sistema de sustitución, pero la atribución tiene tanto arraigo que el nombre de éste método de sustitución ha quedado para los anales de la historia). Otro de los métodos criptográficos utilizados por los griegos fue la escitala espartana, un método de transposición basado en un cilindro que servía como clave en el que se enrollaba el mensaje para poder cifrar y descifrar. La máquina Enigma utilizada por los alemanes durante la II Guerra Mundial

Las dos técnicas más sencillas de cifrado, en la criptografía clásica, son la sustitución (que supone el cambio de significado de los elementos básicos del mensaje -las letras, los dígitos o los símbolos-) y la transposición (que supone una reordenación de los mismos).

Existen dos grandes grupos de cifrado: los algoritmos que utilizan una única clave tanto en el proceso de cifrado como en el de descifrado, y los que utilizan una clave para cifrar mensajes y una clave distinta para descifrarlos. Los primeros se denominan cifrados simétricos o de clave simétrica y son la base de los algoritmos de cifrado clásico. Los segundos se denominan cifrados asimétricos, de clave asimétrica o de clave pública y clave privada y forman el núcleo de las técnicas de cifrado modernas.

Para cualquier tipo de cifrado, el método de ataque más simple es el ataque por fuerza bruta (probando una por una cada posible clave). La longitud de clave determina el número posible de claves, y por tanto la factibilidad del ataque.

A continuación presentamos algunos algoritmos de cifrado así como su origen y uso.

### 2.3.3 MD5, DES, RSA, DSA, IDEA y RC4

Message-Digest Algorithm (MD5)

En criptografía, MD5 (acrónimo de Message-Digest Algorithm 5, Algoritmo de Resumen del Mensaje 5) es un algoritmo de reducción criptográfico de 128 bits desarrollado por el profesor Ronald Rivest del MIT (Massachusetts Institute of Technology, Instituto Tecnológico de Masachusets). Fue desarrollado en 1991 como reemplazo del algoritmo MD4 después de que Hans Dobbertin descubriese su debilidad. A pesar de haber sido considerado criptográficamente seguro en un principio, ciertas investigaciones han revelado vulnerabilidades que hacen cuestionable el uso futuro del MD5. En agosto del 2004, Xiaoyun Wang, Dengguo Feng, Xuejia Lai y Hongbo Yu anunciaron el descubrimiento de colisiones de hash para MD5. Su ataque se consumó en una hora de cálculo con un clúster IBM P690. Aunque dicho ataque era analítico, el tamaño del hash (128 bits) es lo suficientemente pequeño como para que resulte vulnerable frente a ataques de 'fuerza.

Debido al descubrimiento de métodos sencillos para generar colisiones de hash, muchos investigadores recomiendan su sustitución por algoritmos alternativos tales como SHA-1 o RIPEMD-160.

Data Encryption Standar DES

En 1972, el Instituto Nacional de Estándares y Tecnologías NIST por sus iniciales en ingles, decidió que se necesitaba un fuerte algoritmo de cifrado para proteger la información no clasificada. Se requería que el algoritmo fuese barato, ampliamente disponible y muy seguro. En 1974 IBM proporcionó el algoritmo Lucifer, el cual cumplió con la mayoría de los requerimientos de diseño del NIST.

El NIST apoyado por la Agencia Nacional de Seguridad de los Estados Unidos, también conocida como NSA por sus iniciales en ingles, evaluaron la fortaleza de Lucifer. Poco tiempo después de la distribución selecta del algoritmo entre investigadores y personal de inteligencia de la NSA, tanto la longitud de la llave como parte del algoritmo fue cambiado. Originalmente la longitud de la llave era de 128 bits y esta fue cambiada por una llave de 56 bits de longitud, además algunas otras partes del algoritmo (como los bloques de permutaciones) fueron cambiadas arbitrariamente, lo que presupuso el debilitamiento del algoritmo. Este comportamiento provocó que se sospechara de un reacomodo en el algoritmo con el fin de hacer descifrable cualquier información por parte de la NSA con la ayuda de una llave universal o un método inverso de cifrado.

La modificación del algoritmo Lucifer fue adoptada por el NIST como un estándar federal el 23 de noviembre de 1976. Su nombre fue cambiado por el de Cifrado Estándar de Datos ó DES. Las especificaciones del algoritmo se publicaron en enero de 1977 y con el respaldo del gobierno estadounidense, se convirtió rápidamente en un algoritmo de cifrado utilizado ampliamente.

Desafortunadamente, a través del tiempo se presentaron ataques de fuerza bruta cada vez más sofisticados y poderosos que redujeron significativamente el tiempo necesario para descifrar los datos protegidos por DES. Aúnado a esto, el aumento en las capacidades computacionales y velocidades las nuevas computadoras hicieron evidente que las llaves de 56 bits de longitud, simplemente ya no eran lo suficientemente largas para proporcionar niveles altos de seguridad. Una solución a este problema se la llamada versión 3DES, la cual aplica 3 veces consecutivas el algoritmo DES a una misma información.

Actualmente el NIST ha abandonado al algoritmo DES como estándar y ha adoptado al algoritmo Estándar de Cifrado Avanzado o AES (por sus iniciales en ingles), como el algoritmo de cifrado estándar.

El algoritmo DES, pertenece a la familia de algoritmos de cifrado simétrico, y a pesar de representar problemas de seguridad, es aún ampliamente utilizado, sobre todo por su velocidad de cifrado. Hoy en día este algoritmo es complementado generalmente con algún otro algoritmo de cifrado asimétrico para incrementar la seguridad de los datos protegidos.

RSA

De entre todos los algoritmos asimétricos, quizá RSA sea el más sencillo de comprender e implementar. Sus pares de llaves son duales, por lo que sirve tanto para cifrar como para autenticar. Su nombre proviene de sus tres inventores: Ron Rivest, Adi Shamir y Leonard Adleman.

RSA se basa en la dificultad para factorizar grandes números. Las claves pública y privada se calculan a partir de un número que se obtiene como producto de dos primos grandes. El atacante se enfrentará, si quiere recuperar un texto plano a partir del criptograma y la llave publica, a un problema de factorización muy extenso.

La llave privada se utiliza para crear una firma digital, mientras que la pública sirve para verificarla.

La manera más sencilla de entender el cifrado RSA es recordar dos cuestiones de álgebra básica: primero, todos los números tienen un multiplicador inverso, y segundo, (e^a)^b = e^(ab). El multiplicador inverso de un número "*a*" es un número "*b*" tal que *ab=1*. De este modo, en aritmética regular el multiplicador inverso de 4 es 0,25.

RSA utiliza aritmética modular que sólo opera con números enteros entre 0 y un cierto número *n.* A menudo, se compara con el residuo de una división. La ecuación ab mod n podría expresarse como multiplicar a y b, luego dividir el resultado por n y restituir el resto. Por sorprendente que parezca, muchas de las reglas de la aritmética estándar siguen siendo válidas en este dominio. Los números pueden sumarse, restarse, multiplicarse y generalmente dividirse, y las ecuaciones obedecen a las reglas habituales de conmutación, asociatividad y transitividad.

Para construir un par de llaves para RSA, encuentre dos números primos p y q, su producto es n. Las dos claves, e y d, son números aleatorios elegidos de forma que ed mod [(p-1)(q-1)] = 1. Es decir, d es el multiplicador inverso de e.

El algoritmo funciona porque m^de = m mod n (el motivo de que opere también va más allá del propósito de esta explicación). Para cifrar un mensaje, conviértalo en un número m y calcule m^e mod n. Sólo la persona que conoce d puede descifrarlo calculando (m^e mod n)^d mod n = m^de mod n = m.
Actualmente se considera segura una clave RSA con una longitud de "*n*" de al menos 768 bits, si bien se recomienda el uso de claves no inferiores a 1024 bits. Hasta hace relativamente poco se recomendaban 512 bits, pero en mayo de 1999, Adi Shamir presento el denominado dispositivo Twinkle, un ingenio capaz de factorizar números de manera muy rápida, aprovechando los últimos avances en la optimización de algoritmos específicos para esta tarea. Este dispositivo, aún no construido, podría ser incorporado en microcircuitos de bajo coste y pondría en serio peligro los mensajes cifrados con claves de 512 bits o menos.

Teniendo en cuenta los avances de la tecnología, y suponiendo que el algoritmo RSA no sea roto analíticamente, deberemos escoger la longitud de la clave en función del tiempo que queramos que nuestra información permanezca en secreto. Efectivamente, una clave de 1024 bits parece a todas luces demasiado corta como para proteger información por más de unos pocos años por lo que una longitud 2048 bits es en la actualidad la considerada segura.

Ataque de Intermediario (The man in the middle)

El ataque de intermediario puede darse con cualquier algoritmo asimétrico. Supongamos que A quiere establecer una comunicación con B, y que C quiere espiarla. Cuando A le solicite a B su llave publica KB, C se interpone, obteniendo la clave de B y enviando a A una llave falsa KC creada por él. Cuando A cifre el mensaje, C lo interceptara de nuevo, cifrándolo con su propia llave y empleando KB para descifrarlo y enviarlo a B. Ni A ni B son conscientes de que sus mensajes están siendo interceptados.

La única manera de evitar esto consiste en asegurar a "*A*" que la llave pública que tiene de "*B*" es autentica. Para ello nada mejor que esta este confirmada por un amigo común, que certifique la autenticidad de la llave. En la actualidad existen los llamados anillos de confianza, que permiten certificar la autenticidad de las claves sin necesidad de centralizar el proceso. Por eso se nos recomienda cuando instalamos paquetes como el PGP que guardemos todas las llaves sobre las que tengamos certeza de su autenticidad, y solamente esas.

Para evitar este tipo de ataques en necesario implementar un proceso de intercambio de llaves que garantice que el intercambio de éstas con la seguridad de que no han sido intercambiadas o modificadas, para lo cual se han desarrollado, esquemas en los que las llaves pueden ser firmadas, para confirmar que esas llaves pertenecen a quien dicen ser, pero desgraciadamente este esquema también es vulnerable y es necesario fortalecerlo por medio de entidades que certifiquen o validen la autenticidad de las firmas. Para cuestiones prácticas no es necesario llegar a tales niveles de paranoia y bastará con realizar la transferencia de la llave de manera manual (por ejemplo a través de una memoria USB).

Digital Signature Algorithm (DSA)

El Algoritmo de Firma Digital DSA (por sus iniciales en inglés *Digital Signature Algorithm*) fue publicado por el Instituto Nacional de Tecnología y Estándares (NIST) en el estándar llamado Digital Signature Standard (DSS). DSS fue seleccionado por el NIST con ayuda del NSA (National Security Agency) para ser el estándar de autenticación digital del gobierno de los Estados Unidos a partir de Mayo 19 de 1994.

DSA está basado en el problema de logaritmos discretos y se deriva de sistemas criptográficos propuestos por Schnorr y EIGamal. Sirve para firmar y para cifrar información. Una desventaja de este algoritmo es que requiere mucho más tiempo de cómputo que RSA. Su uso principal es la autenticación.

International Data Encryption Algorithm (IDEA)

Algoritmo de cifrado de datos Internacional o IDEA (por sus iniciales en inglés *International Data Encryption Algorithm*) es un cifrador por bloques diseñado por Xuejia Lai y James L. Massey de la Escuela Politécnica Federal de Zúrich y descrito por primera vez en 1991. Fue un algoritmo propuesto como reemplazo del DES (Data Encryption Standard). IDEA fue una revisión menor de PES (*Proposed Encryption Standard*, del inglés Estándar de Cifrado Propuesto), un algoritmo de cifrado anterior. Originalmente IDEA había sido llamado IPES (*Improved PES*, del inglés PES Mejorado).

IDEA fue diseñado en contrato con la Fundación Hasler, la cual se hizo parte de Ascom-Tech AG. IDEA es libre para uso no comercial, aunque fue patentado y sus patentes se vencerán en 2010 y 2011. El nombre "IDEA" es una marca registrada y está licenciado mundialmente por MediaCrypt.

IDEA fue utilizado como el cifrador simétrico en las primeras versiones de PGP (PGP v2.0) y se lo incorporó luego de que el cifrador original usado en la v1.0 ("Bass-O-Matic") se demostró insegura. Es un algoritmo óptimo en OpenPGP.

IDEA opera con bloques de 64 bits usando una clave de 128 bits y consiste de ocho transformaciones idénticas (cada una llamada un ronda) y una transformación de salida (llamada media ronda). El proceso para cifrar y descifrar es similar. Gran parte de la seguridad de IDEA deriva del intercalado de operaciones de distintos grupos —

adición y multiplicación modular y O-exclusivo (XOR) bit a bit — que son algebraicamente "incompatibles" en cierta forma.

IDEA utiliza tres operaciones en su proceso con las cuales logra la confusión, se realizan con grupos de 16 bits y son:

- Operación O-exclusiva (XOR) bit a bit (indicada con un azul)
- Suma módulo 216 (indicada con un boxplus verde)
- Multiplicación módulo 216+1, donde la palabra nula (0x0000) se interpreta como 216 (indicada con un odot rojo)

RC4

Dentro de la criptografía RC4 ó ARC4 es el sistema de cifrado de flujo Stream cipher más utilizado y se usa en algunos de los protocolos más populares como Transport Layer Security (TLS/SSL) (para proteger el tráfico de Internet) y Wired Equivalent Privacy (WEP) (para añadir seguridad en las redes inalámbricas). RC4 fue excluido enseguida de los estándares de alta seguridad por los criptógrafos y algunos modos de usar el algoritmo de criptografía RC4 lo han llevado a ser un sistema de criptografía muy inseguro, incluyendo su uso WEP. No está recomendado su uso en los nuevos sistemas, sin embargo, algunos sistemas basados en RC4 son lo suficientemente seguros para un uso común.

El algoritmo de criptografía RC4 fue diseñado por Ron Rivest de la RSA Security en el año 1987; su nombre completo es Rivest Cipher 4, teniendo el acrónimo RC un significado alternativo al de Ron's Code utilizado para los algoritmos de cifrado RC2, RC5 y RC6.

Inicialmente el algoritmo era un secreto registrado, pero en septiembre de 1994 una descripción del algoritmo fue enviada anónimamente en una lista de correo de Cypherpunks. En seguida pasó al grupo de correo sci.crypt y de ahí fue publicado en numerosos sitos de Internet. Debido al conocimiento del algoritmo, éste dejó de ser un secreto registrado. Sin embargo RC4 aún es una marca registrada. Actualmente la implementación no oficial de RC4 es legal, pero no puede ser utilizada con el nombre de RC4. Por este motivo, y con el fin de evitar problemas legales a raíz de la marca registrada, a menudo podemos verlo nombrado como ARCFOUR, ARC4 o Alleged-RC4. RSA Security nunca ha liberado el algoritmo de su RC4.

RC4 es parte de los protocolos de cifrado más comunes como WEP, WPA para tarjetas wireless y TLS. Entre los factores principales que han ayudado a que RC4 esté en un rango tan amplio de aplicaciones son su increíble velocidad y simplicidad. La implementación tanto en software como en hardware es muy sencilla de desarrollar y son muy pocos los recursos necesarios para obtener un rendimiento eficiente de ARC4.

### 2.3.4 Mecanismos de par de llaves

Un algoritmo asimétrico como por ejemplo RSA puede ser utilizado para generar comunicación cifrada por si sólo, sin embargo en la práctica, este tipo de algoritmo se utiliza sólo para cifrar la llave de sesión generada por un segundo algoritmo (generalmente simétrico). El cifrado de par de llaves es utilizado para generar autenticación de alto nivel.

La comunicación a través de mecanismos de par de llaves (llave privada /llave pública) se realiza generalmente a nivel de máquinas no de usuarios. La autenticación de usuarios se lleva acabo de manera cifrada por medio de la contraseña una vez establecida la autenticación y comunicación de máquina a máquina.

En general la comunicación se establece de la siguiente manera (obviando el componente de cifrado de datos llevado acabo por el segundo algoritmo, como ya mencionamos antes, generalmente llevado acabo por un algoritmo simétrico):

1. La máquina "*A*" o equipo que desea establecer la comunicación con la máquina "*B*", solicita que la máquina "*B*" le envíe su llave pública.
2. La máquina "*B*" envía su llave pública a la máquina "*A*".
3. Por medio de la llave pública de "*B*", la máquina "*A*" cifra la llave maestra del algoritmo simétrico (cualquiera que este sea) con el que se cifrarán los datos a enviar a la máquina "*B*".
4. Los datos cifrados que la máquina "*A*" envía a la máquina "*B*", sólo pueden ser descifrados mediante la utilización de la llave privada de la máquina B.
5. La máquina "*B*" descifra los datos enviados por la máquina "*A*" utilizando su llave privada.
6. Al contestar la máquina "*B*", esta cifra los datos por medio de la llave privada "*B*".
7. Los datos que recibe la máquina "*A*" son descifrados por medio de la llave pública "*B*".

Para el caso de cifrado y autenticación a nivel de usuario, los pasos no cambian mucho, pero el par de llaves tiene que ser generado manualmente por el usuario y compartido con el usuario con el que desea establecer comunicación cifrada.

Este proceso se muestra de manera gráfica en las figuras 2.3, 2.4, 2.5, 2.6 y 2.7

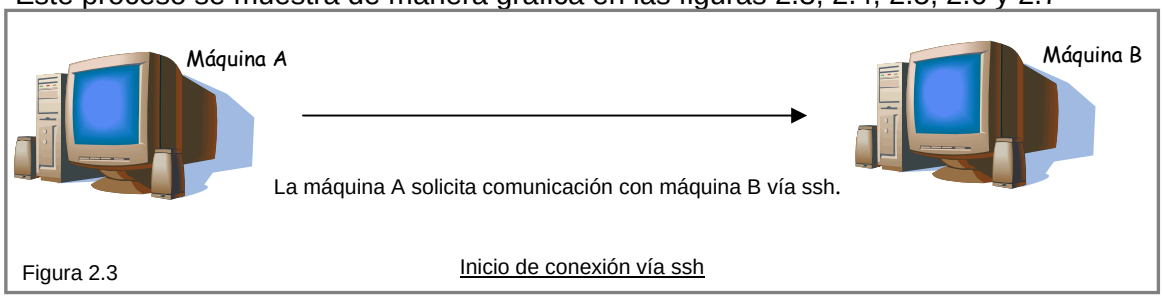


Figura 2.3 Inicio de conexión vía ssh

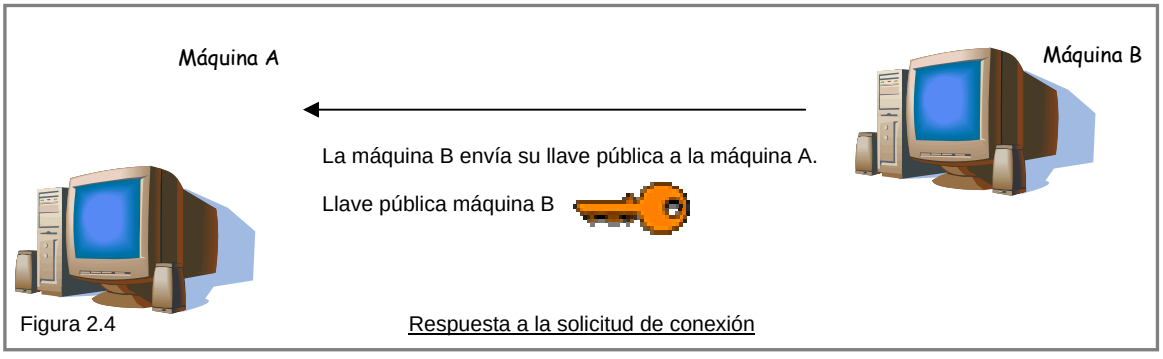


Figura 2.4 Respuesta a la solicitud de conexión

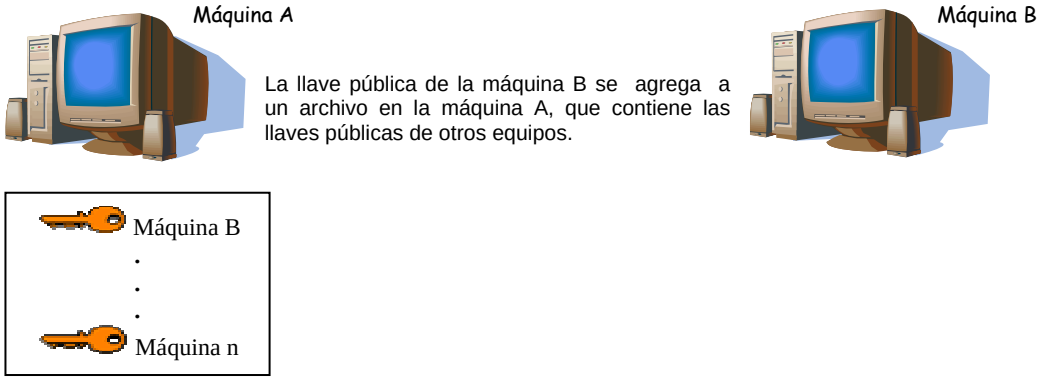


Figura 2.5 Registro de llave pública de la máquina B

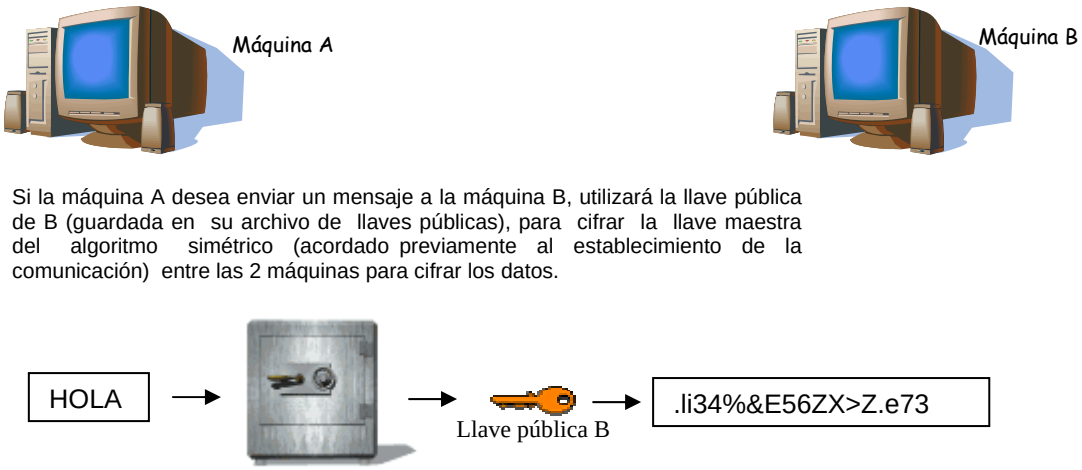


Figura 2.6 Envío de datos cifrados desde A hacia B

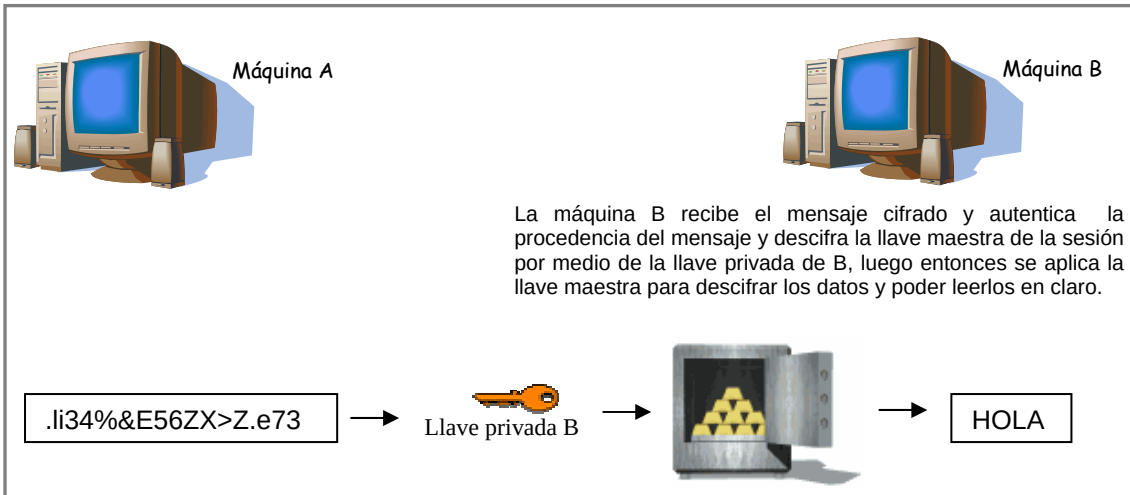


Figura 2.7 Descifrado de datos en B

### 2.3.5 Secure Sokect Layer (SSL)

Historia y desarrollo

El protocolo SSL es un sistema diseñado y propuesto por Netscape Communications Corporation para proporcionar autenticación y privacidad de la información entre extremos sobre Internet mediante el uso de criptografía. SSL versión 3.0 se publicó en 1996, que más tarde sirvió como base para desarrollar TLS versión 1.0. Visa, MasterCard, American Express y muchas de las principales instituciones financieras han aprobado SSL para el comercio sobre Internet.

Se encuentra en la pila OSI entre los niveles de TCP/IP y de los protocolos HTTP, FTP, SMTP, etc. Proporciona sus servicios de seguridad cifrando los datos intercambiados entre el servidor y el cliente con un algoritmo de cifrado simétrico, típicamente el RC4 o IDEA, y cifrando la llave de sesión de RC4 o IDEA mediante un algoritmo de cifrado de llave pública, típicamente el RSA. La llave de sesión es la que se utiliza para cifrar los datos que vienen del y van al servidor seguro. Se genera una llave de sesión distinta para cada transacción, lo cual permite que aunque sea reventada por un atacante en una transacción dada, no sirva para descifrar futuras transacciones. MD5 se usa como algoritmo de hash.

Proporciona cifrado de datos, autenticación de servidores, integridad de mensajes y, opcionalmente, autenticación de cliente para conexiones TCP/IP.

Cuando el cliente pide al servidor seguro una comunicación segura, el servidor abre un puerto cifrado, gestionado por un software llamado Protocolo SSL Record, situado encima de TCP. Será el software de alto nivel, Protocolo SSL Handshake, quien utilice el Protocolo SSL Record y el puerto abierto para comunicarse de forma segura con el cliente.

El Protocolo SSL Handshake

Durante el protocolo SSL Handshake, el cliente y el servidor intercambian una serie de mensajes para negociar las mejoras de seguridad. Este protocolo sigue las siguientes fases (de manera muy resumida):

- La fase Hola, usada para ponerse de acuerdo sobre el conjunto de algoritmos para mantener la intimidad y para la autenticación.
- La fase de intercambio de llaves, en la que intercambia información
- Sobre las llaves, de modo que al final ambas partes comparten una llave maestra.
- La fase de producción de llave de sesión, que será la usada para cifrar los datos intercambiados.
- La fase de verificación del servidor, presente sólo cuando se usa RS como algoritmo de intercambio de llaves, y sirve para que el cliente autentique al servidor.
- La fase de autenticación del cliente, en la que el servidor solicita al cliente un certificado X.509 (si es necesaria la autenticación de cliente). Por último, la fase de fin, que indica que ya se puede comenzar la sesión segura.

### 2.3.6 Aplicación SSH

SSH (Secure SHell) es un programa que sirve para acceder de forma segura a máquinas remotas a través de una red. Permite la ejecución y despliegue de

resultados de línea de comando y también puede redirigir el tráfico de X para poder ejecutar programas gráficos.

Además de la conexión a otras máquinas, SSH nos permite copiar datos de forma segura (tanto archivos aislados como simular sesiones FTP cifradas), gestionar claves RSA para no escribir claves al conectarse

SSH trabaja de forma similar a como se hace con telnet. La diferencia principal es que SSH usa técnicas de cifrado que hacen que la información que viaja por el medio de comunicación vaya de manera no legible y ninguna tercera persona pueda descubrir el usuario y contraseña de la conexión ni lo que se escribe durante toda la sesión.

Al principio sólo existían los r-commands, que eran los basados en el programa rlogin, el cual funciona de una forma similar a telnet.

La primera versión del protocolo y el programa eran libres y los creó un finlandés llamado Tatu Ylönen, pero su licencia fue cambiando y terminó apareciendo la compañía `SSH Communications Security', que lo ofrecía gratuitamente para uso doméstico y académico, pero exigía el pago a otras empresas. En el año 1997 (dos años después de que se creara la primera versión) se propuso como borrador en la IETF.

A principios de 1999 se empezó a escribir una versión que se convertiría en la implementación libre por excelencia, la de OpenBSD, llamada OpenSSH.

## 2.4 Motor de bases de datos

Al igual que el caso anterior, es necesario que el motor de bases de datos sea capaz de ejecutarse dentro de entornos Unix o tipo Unix. Sin embargo en este punto también utilizaremos el costo como discriminador para la elección de los posibles motores de bases de datos.

Derivado de lo anterior quedan como candidatos para la base de datos del sistema de monitoreo de servidores Unix los motores de bases de datos:

- PostGreSQL.
- MySQL.

Ambos PostGreSQL y MySQL pueden trabajar en entornos Unix o tipo Unix, además de estar disponibles libremente a través de licencias tipo BSD y GNU GPL respectivamente, lo que de manera general garantiza su uso de forma libre siempre y cuando las aplicaciones de desarrolladas entren bajo el mismo tipo de licenciamiento.

PostGreSQL

PostgreSQL es un servidor de base de datos objeto relacional libre, liberado bajo la licencia BSD. Como muchos otros proyectos open source, el desarrollo de PostgreSQL no es manejado por una sola compañía sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales las cuales trabajan en su desarrollo, dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

## Historia

PostgreSQL ha tenido una larga evolución, comenzando con el proyecto Ingres en la Universidad de Berkeley. Este proyecto, liderado por Michael Stonebraker, fue uno de los primeros intentos en implementar un motor de base de datos relacional. Después de haber trabajado un largo tiempo en Ingres y de haber tenido una experiencia comercial con el mismo, Michael decidió volver a la Universidad para trabajar en un nuevo proyecto de Ingres, dicho proyecto fue llamado post-ingres o simplemente POSTGRES.

En proyecto post-ingres pretendía resolver los problemas con el modelo de base de datos relacional que habían sido aclarados a comienzos de los años 1980. El principal de estos problemas era la incapacidad del modelo relacional de comprender "tipos", es decir, combinaciones de datos simples que conforman una única unidad. Actualmente estos son llamados objetos. Se esforzaron en introducir la menor cantidad posible de funcionalidades para completar el soporte de tipos. Estas funcionalidades incluían la habilidad de definir tipos, pero también la habilidad de describir relaciones - las cuales hasta ese momento eran ampliamente utilizadas pero mantenidas completamente por el usuario. En POSTGRES la base de datos "comprendía" las relaciones y podía obtener información de tablas relacionadas utilizando reglas.

Después de que el proyecto POSTGRES terminara, Andrew Yu y Jolly Chen, comenzaron a trabajar sobre el código de POSTGRES, esto fue posible dado que POSTGRES estaba licenciado bajo la BSD, y lo primero que hicieron fue añadir soporte para el lenguaje SQL a POSTGRES, dado que anteriormente contaba con su propio lenguaje de consultas, creando así el sistema al cual denominaron Postgres95.

Para el año 1996 se unen al proyecto Marc Fournier, Bruce Momjian y Vadim B. Mikheev quienes comienzan a trabajar para estabilizar el código de Postgres95.

En el año 1996 deciden cambiar el nombre de Postgres95 de tal modo que refleje la característica del lenguaje SQL y lo terminan llamando PostgreSQL.

Con el pasar del tiempo muchos desarrolladores entusiastas de los motores de base de datos se unieron al proyecto y entre todos comenzaron a incorporar muchas características al motor.

## Características

### Alta concurrencia

Mediante un sistema denominado MVCC (Acceso concurrente multiversión, por sus siglas en inglés) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo commit. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases de datos, eliminando la necesidad del uso de bloqueos explícitos.

### Amplia variedad de tipos nativos

PostgreSQL provee nativamente soporte para:

- Números de precisión arbitraria.

- Texto de largo ilimitado.
- Figuras geométricas (con una variedad de funciones asociadas)
- Direcciones Ip (IPv4 e IPv6).
- Bloques de direcciones estilo CIDR.
- Direcciones MAC.
- Arreglos.

Adicionalmente los usuarios pueden crear sus propios tipos de datos, los que pueden ser por completo indexables gracias a la infraestructura GiST de PostgreSQL. Algunos ejemplos son los tipos de datos GIS creados por el proyecto PostGIS.

Soporte para claves foráneas y disparadores o triggers.

Las funciones pueden ser definidas para ejecutarse con los derechos del usuario ejecutor o con los derechos de un usuario previamente definido. El concepto de funciones, en otros DBMS, son muchas veces referidas como "procedimientos almacenados" (*stored procedures* en inglés).

MySQL

Es un sistema de gestión de bases de datos relacional, multihilo y multiusuario de software libre en un esquema de licenciamiento dual.

Por un lado se ofrece bajo licencia GNU GPL, pero, empresas que quieran incorporarlo en productos privativos pueden comprar una licencia que les permita ese uso. Está desarrollado en su mayor parte en ANSI C.

Al contrario de proyectos como el Apache, donde el software es desarrollado por una comunidad pública, y el copyright o derechos de autor del código de MySQL está en poder del autor individual, MySQL es propiedad y está patrocinado por una empresa privada, que posee los derechos de la mayor parte del código.

Esto es lo que posibilita el esquema de licenciamiento anteriormente mencionado. Además de la venta de licencias, la compañía ofrece soporte y servicios. MySQL AB fue fundado por David Axmark, Allan Larsson y Michael Widenius.

Historia

SQL (Lenguaje de Consulta Estructurado) fue comercializado por primera vez en 1981 por IBM, el cual fue presentado en ANSI y desde entonces ha sido considerado el estándar para las bases de datos relacionales. Desde 1986, el estándar SQL ha aparecido en diferentes versiones como por ejemplo: SQL:99, SQL:2003. MySQL es una idea original de la empresa open source MySQL AB establecida inicialmente en Suecia en 1995 y cuyos fundadores son David Axmakr, Allan Larsson, y Michael Monty Widenius. El objetivo que persigue esta empresa consiste en que MySQL cumpla el estándar SQL, pero sin sacrificar velocidad, fiabilidad o capacidad de uso.

Michael Widenius en la década de los 90 trató de usar mSQL para conectar tablas usando rutinas de bajo nivel ISAM, sin embargo, mSQL no era lo suficientemente rápido y flexible para sus necesidades. Esto lo llevó a crear una Interfaz de Programación de Aplicaciones o API (por sus iniciales en inglés *Application Programming Interface*) de SQL denominada MySQL para bases de datos, muy similar a la de mSQL pero más portable.

La procedencia del nombre de MySQL no es clara. Desde hace más de 10 años, las herramientas han mantenido el prefijo My. También, se cree que tiene relación con el nombre de la hija del cofundador Monty Widenius que se llama My.

Por otro lado el nombre del delfín de MySQL es Sakila y fue seleccionado por lo fundadores de MySQL AB en el concurso “Name the Dolphin”.

MySQL es ampliamente utilizado en aplicaciones web, sobre plataformas Unix, Linux y Windows a través la aplicación conjunta Apache-MySQL-PHP. Su popularidad dentro de las aplicaciones Web, esta muy ligada a PHP. MySQL es una base de datos muy rápida en la lectura cuando utiliza el motor no transaccional MyISAM, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación de datos. En aplicaciones Web hay baja concurrencia en la modificación de datos y en cambio el entorno es intensivo en la lectura de datos, lo que hace a MySQL ideal para este tipo de entornos.

MySQL funciona sobre múltiples plataformas, incluyendo AIX, BSD, FreeBSD, HP-UX, GNU/Linux, Mac OS X, NetBSD, Novell Netware, OpenBSD, OS/2 Warp, QNX, SGI IRIX, Solaris, SunOS, SCO OpenServer, SCO UnixWare, Tru64, Windows 95m Windows 98, Windows NT, Windows 2000, Windows XP, Windows Vista y otras versiones de Windows. También existe MySQL para OpenVMS.

Características

- Un amplio subconjunto de ANSI SQL 99, y varias extensiones.
- Soporte a multiplataforma.
- Procedimientos almacenados.
- Triggers.
- Cursores.
- Vistas actualizables.
- Soporte a VARCHAR.
- INFORMATION_SCHEMA.
- Modo Strict.
- Soporte X/Open XA de transacciones distribuidas; transacción en dos fases como pare de esto, utilizando el motor InnoDB de Oracle.
- Motores de almacenamiento independientes (MyISAM para lecturas rápidas, InnoDB para transacciones e integridad referencial).
- Transacciones con los motores de almacenamiento InnoDB, BDB y Cluster; puntos de recuperación (savepints) con InnoDB.
- Soporte para SSL.
- Query caching.
- Sub-SELECTs (o SELECTs anidados).
- Replicación con un servidor maestro por esclavo y muchos esclavos por maestro.
- Indexado y búsqueda de campos de texto completos usando el motor de almacenamiento MyISAM.
- Bibliotecas de bases de datos embebidas.
- Soporte completo para unicode.

Características adicionales

- Usa GNU Automake, Autoconf y Libtool para portabilidad.
- Uso de multihilos mediante hilos del kernel.

- Usa tablas en disco b-tree para búsquedas rápidas con compresión de índice
- Tablas hash en memoria temporales.
- El código MySQL se prueba con Purify (un detector comercial de memoria perdida), así como con Valgrind, una herramienta GPL.
- Completo soporte para operadores y funciones en cláusulas SELECT y WHERE.
- Completo soporte para cláusulas Group by y Order by.
- Seguridad: Ofrece un sistema de contraseñas y privilegios seguro mediante verificación basada en el host y el tráfico de contraseñas esta cifrado.
- Soporta gran cantidad de datos. MySQL Server soporta bases de datos hasta de 50 millones de registros.
- Se permiten hasta 64 índices por tabla.
- Cada índice puede consistir desde 1 hasta 16 columnas o partes de columnas El máximo ancho de límite son 1000 bytes.
- Los clientes y servidores Windows se conectan utilizando memoria compartida.
- MySQL contiene su propio paquete de pruebas de rendimiento proporcionado con el código fuente de la distribución de MySQL.
- Los clientes se conectan al servidor MySQL usando sockets TCP/IP en cualquier plataforma. En sistemas Windows se pueden conectar utilizando named pipes.

Características distintivas

Las siguientes características son implementadas únicamente por MySQL:

- Múltiples motores de almacenamiento (MyISAM, Merge, InnoDB, BDB, Memory/heap, MySQL cluster, Federada, Archivo, CSV y Blackhole), permitiendo al usuario escoger la que sea más adecuada para cada tabla de la base de datos.
- Agrupación de transacciones, reuniendo múltiples transacciones de varias conexiones para incrementar el número de transacciones por segundo.

Ventajas de PostGreSQL

Las características positivas más representativas de esta opción son:

1. Posee una gran escalabilidad. Es capaz de ajustarse al número de CPU's y a la cantidad de memoria que posee el sistema de forma óptima, haciéndole capaz de soportar una mayor cantidad de peticiones simultáneas de manera correcta (estadísticamente hasta el triple de carga de lo que soporta MySQL).
2. Implementa el uso de rollback's, subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz, y ofreciendo soluciones en campos en los que MySQL no podría.
3. Tiene la capacidad de comprobar la integridad referencial, así como también la de almacenar procedimientos en la propia base de datos, equiparándola con gestores de bases de datos de alto nivel, como puede ser Oracle.

Desventajas de PostGreSQL

Y sus inconvenientes más representativos son:

1. Consume gran cantidad de recursos.
2. Tiene un límite de 8K por fila, aunque se puede aumentar a 32K, con una disminución considerable del rendimiento.

3. Es de 2 a 3 veces más lento que MySQL.

Ventajas de MySQL

1. Sin lugar a duda, lo mejor de MySQL es su velocidad a la hora de realizar las operaciones, lo que le hace uno de los gestores que ofrecen mayor rendimiento.
2. Su bajo consumo lo hacen apto para ser ejecutado en una máquina con escasos recursos sin ningún problema.
3. Las utilidades de administración de este gestor son envidiables para muchos de los gestores comerciales existentes, debido a su gran facilidad de configuración e instalación.
4. Tiene una probabilidad muy reducida de corromper los datos, incluso en los casos en los que los errores no se produzcan en el propio gestor, sino en el sistema en el que está.
5. El conjunto de aplicaciones Apache-PHP-MySQL es uno de los más utilizados en Internet.

Desventajas de MySQL

1. Carece de soporte para transacciones, rollback's y subconsultas.
2. El hecho de que no maneje la integridad referencial, hace de este gestor una solución pobre para muchos campos de aplicación, sobre todo para aquellos programadores que provienen de otros gestores que sí que poseen esta característica.
3. No es viable para su uso con grandes bases de datos, a las que se acceda continuamente, ya que no implementa una buena escalabilidad.

Tanto PostGreSQL como MySQL tienen ventajas y desventajas en distintos ámbitos, el punto crucial para decidir cual de las dos se va a utilizar esta más relacionado con las necesidades de uso particulares del sistema, con sus alcances y proyección de crecimiento.

Dado que el sistema de monitoreo de servidores Unix no es una sistema que pueda crecer demasiado ya que eso implicaría que se tiene un número enorme de servidores Unix o tipo Unix que monitorear.

Dependiendo del tamaño de la organización serán los recursos designados para el monitoreo y por tal no es posible establecer que siempre se contara con el recurso idóneo para tal tarea, por lo tanto es deseable que la cantidad de recursos que la bases de datos demande, sea el mínimo necesario para realizar su ejecución de manera óptima.

En base al análisis anterior, MySQL es la opción que representa la mejor solución para la implementación de la base de datos del sistema de monitoreo.

### 2.4.1 Herramienta para bases datos phpmyadmin

PhpMyAdmin es una herramienta libre para la creación, desarrollo y administración de las bases de datos, la cual esta escrita en PHP, con la cual se puede realizar un conjunto de tareas de administración sobre MySQL a través de una interfase Web.

Actualmente puede crear y eliminar bases de datos, crear, eliminar y alterar tablas, borrar, editar y añadir campos, ejecutar cualquier sentencia SQL, administrar claves en campos, administrar privilegios, exportar datos en varios formatos y está disponible en 50 idiomas. Se encuentra disponible bajo la licencia GPL.

PhpMyAdmin puede manejar un servidor de MySQL completo así como una simple base de datos, esto último se logra configurando phpMyAdmin con el usuario correspondiente que pueda leer y escribir sobre las bases de datos deseadas.

Actualmente phpMyAdmin puede:

- Navegar sobre una base de datos, tablas, vistas, campos e índices así como eliminar estos mismos elementos.
- Crear, copiar eliminar renombrar y alterar bases de datos, tablas, campos e índices.
- Dar mantenimiento a servidores, bases de datos y tablas.
- Ejecutar, editar y almacenar cualquier sentencia SQL, incluso consultas en batch.
- Cargar archivos de texto en tablas.
- Crear y leer desbordados de tablas.
- Exportar datos a varios formatos: CSV, XML, PDF, ISO/IEC 26300 – OpenDocument, Word, Excel y formatos de LATEX.
- Administrar múltiples servidores.
- Administrar usuarios y privilegios de MySQL.
- Verificar integridad referencial en tablas MyISAM.
- Crear gráficos PDF de la distribución de la base de datos.
- Buscar de manera global en la base de datos o en un subconjunto de ésta.
- Soporte para tablas InnoDB y llaves foráneas.
- Soporte para mysqli (las extensiones mejoradas de MySQL).
- La interfase se presenta en 54 diferentes lenguajes.
- phpMyAdmin puede realizar compresión Zip, GZip –RFC 1952 o Bzip2.

## 2.5 Herramientas para la programación en Web

Historia de Apache

En Febrero de 1995, el servidor más popular de HTML en la Web era el demonio de http, desarrollado por Rob McCool en el Centro Nacional de Aplicaciones de Supercómputo en la universidad de Illinois, mejor conocido como NCSA por sus iniciales en inglés (*National Center for Supercomputing Applications*). Sin embargo el desarrollo de httpd se estanco después de que Rob dejara la NCSA a mediados de 1994. Durante el transcurso de ese año algunos webmasters desarrollaron sus propias extensiones y correcciones de bugs o errores y consideraron la necesidad de agrupar estas mejoras en una sola distribución, así fue como surgió el grupo fundador de lo que hoy se conoce como Apache.

El grupo original se constituyó por: Brian Behlendorf, Roy T. Fielding, Rob Hartill, David Robinson, Cliff Skolnick, Randy Terbush, Robert S. Thau y Andrew Wilson

Basados en el httpd 1.3 de NCSA, se agregaron todas las correcciones y mejoras, lo que trajo consigo la liberación oficial de la primera versión de Apache, Apache 0.6.2 en abril de 1995.

Se dice que el nombre de Apache proviene del juego de palabras “a patch” o un parche, debido a que muchas de las mejoras originales fueron parches a la estructura original. Además de que Behelendorf consideró necesario que el nombre tuviera la connotación de algo que es firme y enérgico, y la tribu Apache fue la última en rendirse ante el gobierno de EEUU.

El servidor de Apache fue todo un éxito, pero el grupo sabia que el código base necesitaba un reacondicionamiento y rediseño general. Entre Mayo y Junio de ese año, Robert Thau diseñó una nueva arquitectura de servidor, la cual era modular y junto con otras mejoras en forma de módulos desarrolladas por el grupo de trabajo, finalmente el primero de Diciembre de 1995 salio a la luz Apache 1.0.

En menos de un año, Apache se convirtió en el servidor número 1 en la Internet, puesto que ha mantenido hasta el día de hoy.

Ventajas

El servidor de HTTP Apache es un software de código abierto principalmente para plataformas Unix (BSD, GUN/Linux, etc.), Windows, Macintosh y otras que implementan el protocolo HTTP/1.1.

Apache presenta entre otras características mensajes de error altamente configurables, bases de datos de autenticación y negociado de contenido.

Las características generales de apache son:

- Modular.
- Open Source.
- Multi-plataforma.
- Extensible.
- Popular (fácil conseguir ayuda/soporte).
- Gratuito.

Hojas de estilo (CSS)

Las hojas de estilo o CSS por sus siglas en inglés (*Cascade Style Sheets*), son hoy en día un elemento esencial de las páginas web, éstas pueden ahorrar mucho tiempo y trabajo, las hojas de estilo definen, cómo los elementos de HTML serán desplegados, al igual que la etiqueta de font y el atributo de color en el HTML normal. Las hojas de estilo se almacenan normalmente en archivos con la extensión .css. Las hojas de estilo nos permiten cambiar la apariencia y la distribución de toda una página web tan solo editando el contenido de un solo documento de CSS.

Las hojas de estilo en su momento significaron un parte aguas, debido a permitieron que los desarrolladores controlaran tanto el estilo como distribución de múltiples páginas web. Las hojas de estilo permiten definir el comportamiento y estilo de diferentes elementos HTML y aplicar este comportamiento a las páginas web tan solo agregando la referencia al documento CSS, lo que permite además realizar cambios globales simplemente modificando la hoja de estilo correspondiente.

# Capítulo III

# Monitoreo

## 3.1 Monitoreo

El monitoreo por definición es la acción repetitiva de observar, muestrear y analizar parámetros predefinidos, con el objeto de determinar si dichos datos o información recolectada (ya sea en manera individual o en conjunto) se encuentra dentro de un rango definido por métricas especificas y relacionadas a los datos recolectados y que permiten a su vez no sólo ubicar esos datos dentro de un rango cuantitativo sino también cualitativo.

### 3.1.1 Enfoques de monitoreo

En general el monitoreo puede clasificarse en monitoreo activo o pasivo, ambos tienen ventajas y desventajas respecto uno de otro. Muchas de sus características son mutuamente excluyentes, pero algunos resultados pueden ser obtenidos por cualquiera de los dos métodos. Aunque es posible utilizar uno u otro por separado, es en conjunto cuando se obtienen resultados más completos.

### 3.1.2 Monitoreo Activo

El monitoreo activo de redes, se basa en la capacidad de inyectar paquetes de prueba en la red o enviar paquetes a servidores y aplicaciones y posteriormente llevar un seguimiento y una medición de las respuestas a estos paquetes de prueba. El volumen de tráfico extra necesario para obtener resultados significativos no es considerable, debido a que en la mayoría de los casos los parámetros son artificiales, es decir, el volumen y otros parámetros del tráfico introducido son ajustables.

Ejemplos monitoreo activo son:

- El monitoreo por ICMP, con el cual se puede detectar problemas de red analizando el retardo o la pérdida de paquetes para determinar la disponibilidad del equipo, dispositivo o de la red misma.
- El monitoreo basado en TCP con el cual se puede medir la tasa de transferencia y diagnosticar problemas a nivel de aplicación.
- El monitoreo basado en UDP al cuantificar la pérdida de paquetes en un sentido (one-way) por medio del análisis de las características de RTT (round-trip-times).

Por otro lado el monitoreo activo proporciona control explicito de la generación de paquetes para escenarios de medición. Esto incluye control de la naturaleza de generación de tráfico, las técnicas de muestreo, los tiempos, la frecuencia, la calendarización, el tamaño y tipos de paquetes (con el propósito de emular diferentes aplicaciones), calidad estadística, la ruta y las funciones elegidas para ser monitoreadas.

El monitoreo activo no se circunscribe sólo al monitoreo de parámetros y variables de red sino que incluye el monitoreo de elementos internos a los dispositivos o servidores de los cuales se extraen los valores de estos parámetros y se envían a sistemas centrales en los que se analiza esa información.

### 3.1.3 Monitoreo Pasivo

Podría decirse que en la práctica el monitoreo pasivo es aquel que no perturba al objeto del monitoreo y por consiguiente fotografía al objeto de estudio en sus condiciones habitúales de comportamiento sin introducir nuevas variables, parámetros o modificar los valores sujetos de análisis del objeto monitoreado.

En el entorno de redes computacionales, este concepto se refiere a la obtención de información del objeto de monitoreado sin introducir tráfico adicional a la red o procesos adicionales al servidor o dispositivo monitoreado.

En el caso de los dispositivos interconectados a una red, los únicos datos que se pueden obtener son los que el mismo dispositivo envía o recibe, en respuesta a peticiones o notificaciones que otros dispositivos o hosts conectados a la red le dirigen.

El enfoque de monitoreo pasivo utiliza dispositivos de propósito especial como sniffers o sistemas ínter-construidos en otros dispositivos como switches, ruteadores o en servidores con software especial para analizar y observar el tráfico que atraviesa la red.

Ejemplos de estas técnicas son el Monitoreo Remoto (RMON), el Protocolo Simple de Administración de Red (SNMP) y el netflow.

El monitoreo pasivo es una técnica utilizada para capturar tráfico de una red y generar una copia de dicho tráfico, ya sea por medio puertos span, puertos espejo o por dispositivos network tabs. Una vez que los datos (flujo de datagramas o paquetes) han sido extraídos pueden ser utilizados en diferentes formas.

- Pueden ser analizados por un sniffer.
- Pueden ser examinados por herramientas de flujo de tráfico, proporcionando información de los dispositivos o hosts en la red así como datos de tiempo y distancia entre hosts o dispositivos lo que a su vez permitiría proporcionar una imagen de la estructura de red.
- Pueden ser reensamblados en base a la actividad del usuario final con respecto a aplicaciones en especifico (por ejemplo consultas a bases de datos, mensajes de correo electrónico, etc.) o simplemente con el propósito de facturar cierto tipo de tráfico.

El monitoreo pasivo es de gran ayuda en la resolución de problemas, pero el análisis y la información que de él se obtiene se deriva de datos capturados previamente, por lo que los errores o problemas sólo pueden ser detectados una vez que estos se han presentado.

Aún y cuando el monitoreo pasivo es muy valioso en la solución de problemas de red, es limitado en la emulación de escenarios de error y dado que se requiere de la lectura de todos los paquetes en la red y existen problemas de seguridad y confidencialidad asociados a él.

## 3.2 Políticas de Cómputo

Conforme el tiempo avanza es cada vez más un hecho que las cuestiones de seguridad y el monitoreo de los sistemas y recursos computacionales, es no

solamente importante, sino indispensable para garantizar, no sólo la continuidad de la operación, sino la continuidad de la operación de manera segura.

Por ejemplo, de nada sirve garantizar la continuidad del proceso de desarrollo de un nuevo dispositivo si en el camino no hemos garantizado su secrecía y dicho desarrollo ha sido robado por un tercero y liberado antes de tiempo o como propio. De nada sirve tener un sistema muy bueno, si los recursos que lo sustentan están dañados o corruptos y fallan de manera intermitentemente.

Hoy en día las organizaciones, instituciones o empresas regulan gran cantidad de sus actividades y procesos, de igual manera las actividades, procesos o servicios sistematizados son sujetos de normatividad.

Las políticas se establecen para normar un conjunto de acontecimientos posibles circunscritos a una tarea o proceso en particular. Dichas políticas establecen criterios para el uso de los recursos y el comportamiento general al uso de los mismos, ya sean éstos lógicos o físicos. La política debe establecer de manera clara e indubitable las facultades a las que los usuarios (a demás de definir tipos de usuarios y los alcances y perfiles de estos de ser necesario) tendrán derecho, las obligaciones generales (y particulares de así presentarse el caso), y deberán establecer las sanciones (basadas en leyes, normas, acuerdos, circulares o acuerdos internos, todos ellos documentados) a las que él o los infractores se harán acreedores en caso de infringir los lineamientos establecidos por la política.

En relación a los usuarios, sus facultades, obligaciones y sanciones, cabe mencionar que los usuarios con perfil de administradores, también deben estar definidos y delimitados en los puntos antes mencionados.

En el entorno computacional convergen diversas disciplinas o especialidades, por lo que en la mayoría de los casos los administradores para cada una de ellas será diferente, aunque esto último no es obligatorio y bien podemos encontrar casos en los que el administrador de la red LAN, o WAN, también sea el administrador de la seguridad (firewalls o de otro tipo de sistemas o aplicaciones). Aún así los alcances y funciones especificas de éstos deberán estar bien definidos para evitar que las tareas y alcances de uno se traslapen con las del otro. Por ejemplo un punto de la política podría hacer referencia a que sólo el administrador de red o el administrador de seguridad pudieran realizar barridos de puertos y detección de equipos en la red, por lo que un barrido de puertos de manera externa por parte de un administrador de servidores podría bien estar violentando la política.

En entornos de trabajo en donde todo este tipo de normas esta ya definido, es importante que el diseño de cualquier nuevo sistema tenga en cuenta los lineamientos definidos, para evitar así transgredir cualquier comportamiento definido en la política. Por otro lado el sistema de monitoreo de servidores Unix proporciona la ventaja de que los resultados son obtenidos de manera no intrusiva ni por medio de mapeos o barridos de red, sino que esta información se extrae desde dentro del propio servidor y por ello no es necesario permisos de red especiales.

### 3.2.1 ¿Qué monitorear? y ¿para qué?

Se dice que una de las actividades principales de los administradores de sistemas es el monitoreo, pero ¿qué es lo que ese monitoreo comprende o abarca?, la respuesta a esta pregunta va por fuerza acompañada de la pregunta ¿para qué?, la primera pregunta responde o lista los posibles elementos que conforman un monitoreo de

cierto tipo, pero es la segunda pregunta, la que en realidad nos da la pauta para analizar, definir, sustentar y darle forma al monitoreo específico o definir el conjunto de elementos que constituirá ese monitoreo.

Objetos sustantivos, representativos y acordes a los objetivos, son la respuesta la pregunta ¿qué monitorear?.

- Sustantivos: es decir, los objetos propuestos para el monitoreo deben ser capaces de proporcionar información valiosa a través de la cual se pueda establecer de forma directa o bien en base a ellos tener la capacidad de inferir valores cuantitativos y/o cualitativos.
- Representativos: los objetos deben de tener un peso especifico suficientemente alto como para considerarlos determinantes en la valoración de una cuantificación o cualificación.
- Acordes: los objetos deben pertenecer o estar lo suficientemente relacionados a un conjunto de elementos orientados a la definición del objetivo de monitoreo, por ejemplo, no tendría mucho sentido monitorear el número de procesos locales de un usuario en particular en un servidor, si lo que se esta buscando de determinar es si existe un cuello de botella en la red.

Derivado de lo anterior, el conjunto de elementos de monitoreo está directamente relacionado al tipo de resultado que se espera obtener de ellos.

En nuestro caso, el objetivo del monitoreo, es establecer el estado del servidor de una manera segura, no intrusiva y que arroje información valiosa y digerida que nos permita, tanto anticipar eventualidades como reaccionar a estas de manera pronta y así aminorar los posibles impactos negativos en la operación y procesos de negocio.

### 3.2.2 Métricas

En este caso, métrica hace referencia a la definición del rango de valores posibles que se pueden utilizar o generar, la escala a utilizar y el tipo de unidades coherentes al objeto monitoreado.

Las herramientas de monitoreo en general no verifican todos y cada uno de los elementos de un servidor, la selección de los elementos de monitoreo esta directamente relacionada a la función central de la herramienta, es decir, una herramienta de monitoreo de vulnerabilidades utilizará diferentes métodos y métricas que una herramienta de monitoreo de red, y aunque ambas podrían determinar la utilización del ancho de banda de un canal utilizado por cierto dispositivo o servidor, el de monitoreo de vulnerabilidades podría estar interesado en los patrones de comportamiento nocivo y la velocidad o agresividad con el que este esta actuando y el monitoreo de red sólo trataría de localizar al responsable de la sobre carga en el canal.

Es así que los métricas o los parámetros de monitoreo se definen desde la creación de la herramienta, con el propósito de que los datos generados por esta proporcionen información sustancial, valiosa y orientada a un propósito, función o tarea específica.

Como ejemplo de diferentes métricas tenemos:

- Métricas de tráfico de entrada y salida, la cual se puede acusar a través del número de paquetes enviados, paquetes recibidos y paquetes perdidos. Se habla de paquetes y no de datagramas, dado que los datagramas son paquetes fragmentados y lo que nos interesa contar son los paquetes que han

sido reensamblados correctamente y han sido procesados por el dispositivo de red (tarjeta de red) o los que se ha logrado enviar con éxito o la contabilidad de los que se han perdido o colisionado en el medio físico.

- Métricas de utilización de procesador, las cuales nos indican si el procesador cuenta con la capacidad de atender las tareas de procesamiento con la rapidez necesaria.
- Métricas de memoria RAM (física primaria) y métricas de memoria secundaria o memoria swap en las cuales la cota superior es el limite físico de ellas ya sea la capacidad de almacenamiento que de fabrica poseen o el tamaño que se les asigne, respectivamente para cada uno de los dos tipos de memoria. Las escalas de las unidades varían con el tiempo dado que las capacidades de almacenamiento crecen también y hoy en día por ejemplo no es usual escuchar de memorias RAM en medidas en Kilo bytes.

En algunos casos la cuantificación de la utilización de los objetos se especifica en porcentajes. Sin embargo este tipo de medición no siempre describe de manera real es estado de salud de un objeto. Ejemplo de esto son los sistemas de archivos y la memoria RAM, dado que en el caso de un sistema de archivo cuyo tamaño sea de 900 MB con una utilización en porcentaje de 70%, esto no significa que el tamaño sea adecuado para una función específica. Supongamos ahora que el sistema de archivos es /var y que nos encontramos en un sistema en producción con diferentes bases de datos las cuales generan archivos de bitácoras o incrementan los registros a un ritmo promedio de 50 Mb por hora, claramente este sistema de archivos se saturará muy pronto y con ello la operación de la bases de datos y los servicios que de ella dependan.

Es importante analizar el tipo y el tamaño del objeto para así poder definir adecuadamente los valores bajo los cuales las alertas y las alarmas serán disparadas.

### 3.2.3 Definición de alarmas y alertas

Alarma

Por definición alarma proviene de la contracción de “al arma” que es el grito con el que se llamaba a combate, aviso que se daba para advertir de un peligro inminente o del ataque de un enemigo. En computación y entornos de red, se utiliza la palabra alarma para referirse a una situación de peligro que esta sucediendo o esta a punto de suceder de forma inminente. Las alarmas o más bien dicho su detección, es de carácter reactivo y generalmente se configuran para eventos los cuales afectan la continuidad de la operación y por su naturaleza prioritaria requieren atención inmediata.

Las alarmas en un esquema de monitoreo suceden generalmente después de las alertas, aunque dependiendo del tipo de objeto de monitoreo la alarma puede presentarse sin la necesidad de una alerta previa, como ejemplo de esto tenemos los puertos de comunicación, en el primer caso cuando una intrusión se ha logrado de manera satisfactoria, una de las primeras actividades que un atacante realiza, es abrir un puerto de comunicación por el cual tener acceso al sistema sin ser detectado, la aparición de un nuevo puerto en el sistema puede entonces darse sin la necesidad de que una alerta previa se halla reportado. En el caso de la caída de un puerto de comunicación, este puede darse de manera intempestiva, más sin embargo, su caída puede deberse a cuestiones de falta de memoria, en este caso el monitoreo no generará una alerta a causa del puerto si no por memoria. El conocimiento de estas relaciones hasta cierto punto obvias evitaría que el servicio ofrecido a través de este

puerto se suspendiera, es así que el análisis de la alerta de un objeto nos puede llevar a diagnosticar un posible problema en otro objeto de monitoreo.

Alerta

Por definición alerta es la señal que previene del peligro, es el aviso dado a una fuerza militar para que se prepare y este en situación de intervenir en un plazo fijado.

Las alertas en el monitoreo representan el aviso ante una situación o evento que necesita atención con el propósito de que no se convierta en un problema. Las alertas indican que el valor de un parámetro u objeto de monitoreo ha alcanzado niveles fuera del comportamiento normal o de línea base, pero aún no representa una amenaza para la continuidad de la operación. El propósito principal de la alerta es dar aviso para que el proceso de atención a incidentes inicie. Este proceso esta orientado a reducir o regresar al objeto de monitoreo a valores alrededor de los establecidos en la línea base.

## 3.3 Herramientas de monitoreo

El monitoreo de servidores se puede llevar acabo de varias maneras, cada una con sus pros y sus contras. Los sistemas Unix o tipo Unix son sistemas caracterizados por generar un gran número de bitácoras o registros de casi todas sus acciones o eventos, tanto de error como de éxito o meramente estadísticos, además de contar con herramientas de bajo nivel o comandos capaces de explotar la información de estos registros y presentarla de manera legible (ya que en ocasiones los registros o datos se encuentran almacenados en forma binaria o con formatos poco entendibles).

De manera práctica, la ejecución de un conjunto de comandos que proporcionen datos los cuales tengan que ser analizados, ordenados o filtrados o el análisis de las bitácoras para obtener la información puntual, se le denomina monitoreo manual los resultados de esta manera obtenidos tienen la misma validez de los datos obtenidos por herramientas automáticas, la diferencia principal deriva en la capacidad de las herramientas automatizadas de ejecutarse periódicamente y de ordenar los datos obtenidos en formatos legibles y sustanciosos, así como también de reducir el tiempo invertido para esta labor.

Las herramientas de monitoreo son importantes debido a que proporcionan información valiosa todo el tiempo y en mucho menor tiempo. Y aunque en realidad no existe nada que un buen administrador no pueda detectar de igual forma que una herramienta de monitoreo, también es verdad que las tareas de un administrador no se reducen tan sólo al monitoreo, aunado a esto, los administradores son personas y como tales necesitan descansar, lo que implica que no podrán realizar el monitoreo de manera ininterrumpida.

Las aplicaciones de monitoreo son herramientas valiosas para los administradores ya que les ayudan en la detección oportuna de fallas y les permiten utilizar sus conocimientos y creatividad en actividades de mayor provecho para la organización en la que laboran.

### 3.3.1 Características de Nagios

Nagios es un programa de monitoreo de equipos y servicios de red, diseñado para informar de problemas de red. Nagios nació originalmente con el nombre de Netsaint y fue creado y es actualmente mantenido por Ethan Galstad junto con un gran grupo de desarrolladores. Fue diseñado para trabajar sobre Linux, pero también puede trabajar sobre sistemas operativos Unix, tipo Unix.

Según Ethan Galstad en el apartado de FAQs del sitio oficial de Nagios, N.A.G.I.O.S es un acrónimo recursivo de "Nagios Ain't Gonna Insist On Sainthood" o "Nagios no insistirá en la Santidad", el cual es una referencia a su anterior nombre, Netsaint el cual tuvo que ser cambiado por ser similar a un nombre comercial.

Nagios tiene licencia pública general versión 2 de GNU o GNU-GPLv2, publicada por la Fundación de Software Libre (Free Software Fundation).

El demonio de monitoreo se ejecuta de manera intermitente y revisa los servidores y los servicios especificados utilizando plugins externos los cuales regresan información del estado de éstos. Cuando se detectan problemas, el demonio envía notificaciones a los contactos administrativos (estas notificaciones pueden ser enviadas en diferentes formatos: correo electrónico, mensaje instantáneo, SMS, etc.). Información del estado actual, bitácoras históricas y reportes, pueden ser accedidos vía navegador Web.

Características de Nagios

Nagios tiene muchas características, las cuales lo hacen una herramienta de monitoreo muy ponderosa. Algunas de las principales características se listan a continuación:

- Monitoreo de servicios de red como SMTP, POP3, HTTP, NNTP, y detección a través de PING etc.
- Monitoreo de recursos de máquina (carga en el procesador, utilización de disco y memoria, ejecución de procesos etc).
- Monitoreo de factores ambientales como temperatura.
- Diseño simple de módulos, que permite a los usuarios desarrollar de manera fácil sus propias revisiones de máquina y servicios.
- Habilidad para definir jerarquías de máquinas en la red, detectar y distinguir entre máquinas que están apagadas o desconectadas de la red de aquellas que se encuentran inalcanzables.
- Notificaciones a contactos cuando ocurren problemas en las máquinas o en los servicios.
- Escalamiento opcional de las notificaciones a diferentes grupos de contactos.
- Habilidad de definir acciones ante la ocurrencia de eventos en los servicios y así atacar de manera proactiva los problemas y su resolución.
- Soporte para implementar monitoreo redundante y distribuido de servidores.
- Interfase de comandos externos que permite modificaciones en tiempo real al monitoreo y al comportamiento de las notificaciones.
- Retención de el estado de los servicios y de las máquinas después de un reinicio de programa.
- Calendarización de tiempos fuera para las máquinas y los servicios para evitar la notificación de éstos durante el tiempo de no operación.
- Habilidad para el reconocimiento de errores vía interfase web.
- Interfase Web para la visualización del estado actual de la red, notificaciones y problemas históricos.

- Un esquema de autorización simple, que permite restringir lo que los usuarios pueden ver y hacer en la interfase Web.

A diferencia de muchas otras herramientas de monitoreo, Nagios no incluye mecanismos internos para realizar la revisión del estado de las máquinas y de los servicios en la red. En lugar de eso, Nagios utiliza programas externos (llamados plugins) para hacer todo el trabajo sucio.

Los plugins son programas ejecutables compilados o scripts (de Perl, algún intérprete de comandos, etc.) que puedan ser ejecutados desde una línea de comandos para revisar el estado de una máquina o de un servicio en la red.

Nagios ejecutará un plugin cada vez que exista la necesidad de revisar el estado de un servicio o de una máquina, el proceso de manera general es el que sigue: El plugin realiza alguna acción de revisión y luego simplemente regresa el resultado a Nagios. Nagios procesará los resultados que reciba del plugin y tomará las acciones necesarias.

Los plugins actúan como una capa de abstracción entre la lógica de monitoreo presente en el demonio de Nagios y los servicios y máquinas a monitorear, como se muestra en la figura 3.1.

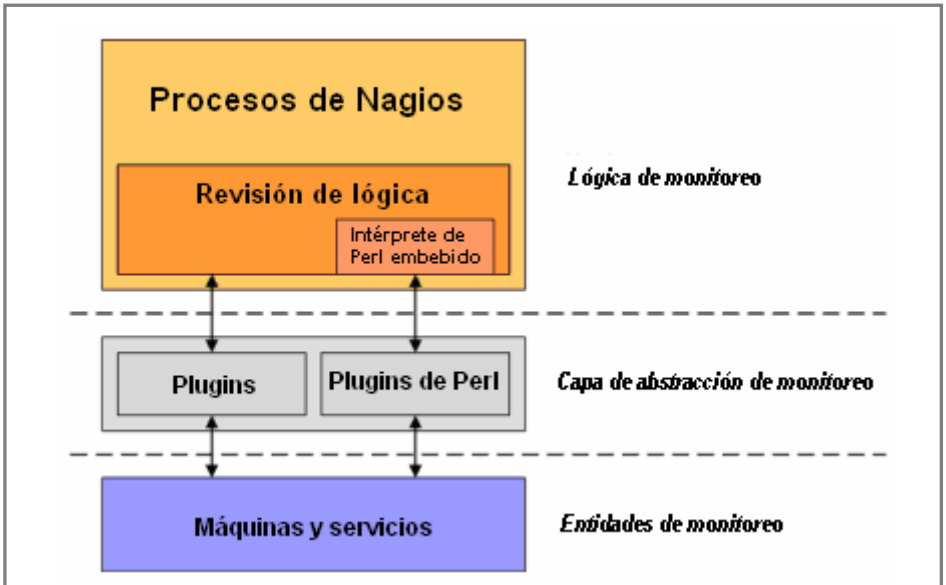


Figura 3.1 Esquema lógico de Nagios

La ventaja de este tipo de arquitectura de plugin es que se puede monitorear cualquier cosa que se pueda uno imaginar. Si uno puede automatizar el proceso de revisar algo, uno lo puede monitorear con Nagios. De hecho se han ya desarrollado muchos plugins para monitorear los recursos básicos, como la carga en el procesador, la utilización de disco, estadísticas por medio de pings, etcétera.

La arquitectura de Nagios en realidad no tiene idea de que es lo que es esta monitoreando. El objeto de monitoreo puede ser el tráfico de red, estadísticas de errores, temperatura, voltaje, CPU, espacio en el disco. Nagios no conoce las especificaciones de lo que se esta monitoreando, él sólo le da seguimiento a los

cambios de estado de esos recursos. Sólo los plugins saben exactamente que es lo que se esta monitoreando y la manera de cómo se realiza este monitoreo.

Existen plugins para diferentes tipos de dispositivos y servicios, incluyendo:

- HTTP, POP3, IMAP, FTP, SSH, DHCP.
- Carga de CPU, utilización de disco, uso de memoria, usuarios actuales.
- Servidores Unix/Linux, Windows y Netware.
- Ruteadores y Switches.
- Etc.

Plugins

Los plugins no se distribuyen junto con Nagios, pero se pueden descargar de la página oficial de plugins de Nagios o desde otras direcciones creadas y mantenidas por usuario de Nagios.

Requerimientos de Nagios

El único requerimiento para ejecutar Nagios es una máquina con Linux (o una variante de Unix), un compilador de C, un servidor http (de preferencia Apache), una base de datos (de preferencia MySQL), y una interfase de red con soporte para TCP/IP, ya que las revisiones de servicios se realizan sobre la red.

Instalación de Nagios

En referencia a la instalación y configuración de esta herramienta, la propia página oficial de Nagios advierte que aunque Nagios es una herramienta muy poderosa y flexible, ésta también puede demandar de mucho trabajo para lograr su configuración y ponerla en operación. Y proporciona los siguientes consejos:

1. Relájese – esto tomará algún tiempo. No espere que la cosas trabajen justo como las pensó de manera inmediata, esto no es tan fácil. Tener Nagios en operación puede involucrar un poco de trabajo, en parte por las opciones que Nagios ofrece y parte porque necesitas saber de ante mano que es lo que deseas monitorear de tu red.
2. La guía rápida de instalación proporcionará los pasos básicos a los nuevos usuarios para que estos puedan poner a Nagios en una configuración básica.
3. Es importante leer la documentación. Nagios puede ser engañoso de configurar aún cuando se tenga una buena idea de que es lo que esta pasando y casi imposible si no se tiene ni idea. Además sugiere reservar los tópicos avanzados de Nagios para cuando se tenga mayor entendimiento de la herramienta.
4. El siguiente punto al que hacen referencia puede ser considerado tanto malo como bueno. Se invita a recibir ayuda a través de las listas de usuarios, lo cual puede ser considerado malo ya que no cuenta con una documentación de errores y soluciones. Pero por otro lado el mecanismo de listas de usuarios es cada vez más utilizado, para resolver problemas particulares de manera rápida.
5. Nagios no proporciona un mecanismos de configuración en línea que permita modificar los parámetros de monitoreo o agregar equipos a través de la interfase Web. Todo este tipo de modificaciones se realizar a través de archivos de configuración en el directorio raíz de instalación.

Nagios proporciona mucha y detallada información acerca del estado de los equipos que monitorea (quizás demasiada información). Un ejemplo de la información que por servicio puede mostrar Nagios es el presentado por la figura 3.2.

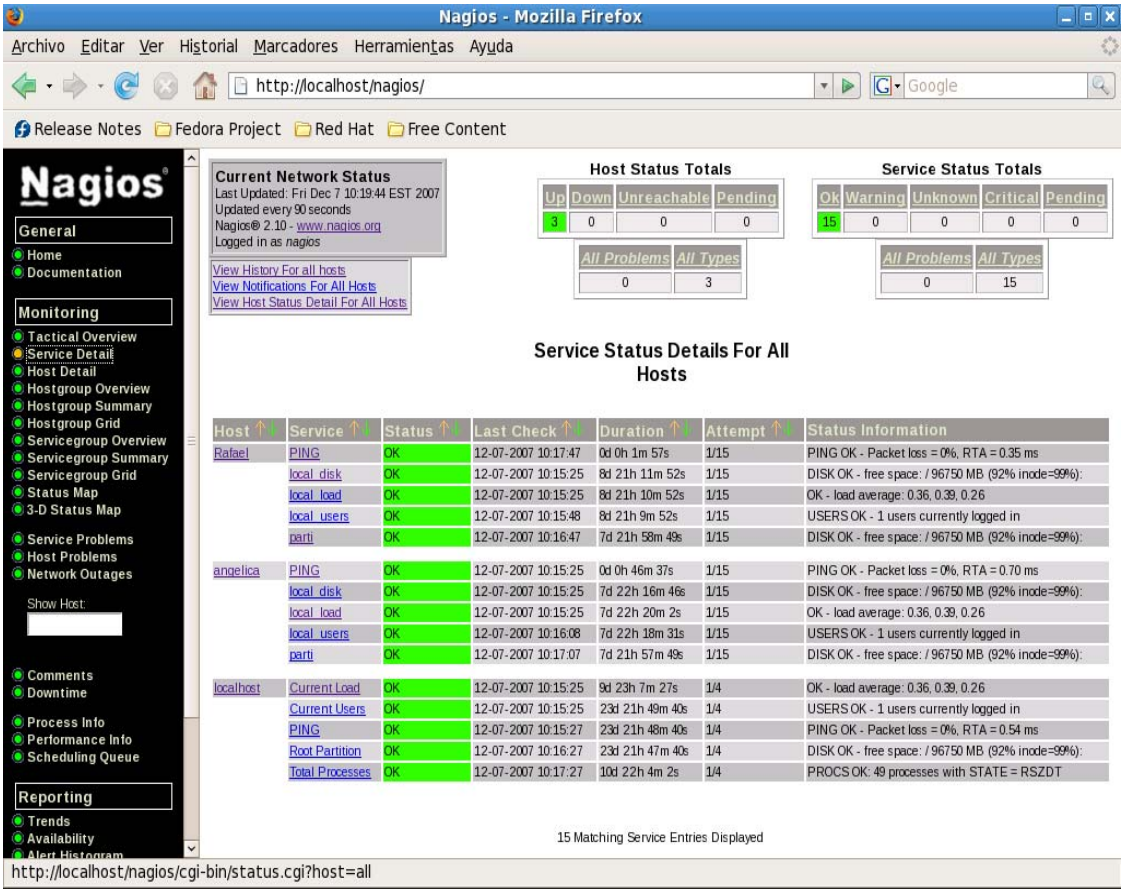


Figura 3.2 Vista de monitoreo de servicios en Nagios

Esta figura muestra los datos monitoreo de 2 equipos y del equipo local. Mostrando de izquierda a derecha (en el bloque central principal), en la primera columna el nombre del equipo (este nombre lo definimos nosotros en los archivos de configuración, así como también definimos los servicios que de él se desean monitorear), en la segunda columna se muestran los servicios, en la tercera se muestra el estado de estos servicios, en al cuarta columna se muestra la fecha en la que se realizó esta revisión, la siguiente columna muestra el tiempo que este servidor a estado activo o encendido (no necesariamente monitoreado ni conectado a red, estos datos son idénticos a los mostrados por el comando uptime en los sistemas operativos Unix o tipo Unix), la siguiente columna muestra el número de intentos necesarios para realizar la revisión y por último la última columna muestra el resultado de lo monitoreado.

En este caso, en la última columna del equipo denominado Rafael, se observa que se lanzaron paquetes de ICMP a través de PING de los cuales no se perdió ninguno y que el promedio transmisión por paquete fue de 0.35 milisegundos, que estado del disco en cuanto a espacio se considera aceptable, que la carga promedio también es aceptable y nos da tres lecturas de las cuales se asume (aunque no se indica en ningún lado) que son las mismas lecturas que un comando como uptime proporcionaría, es decir, la primera lectura representa la carga promedio al minuto, la segunda lectura representa la carga promedio a de los 5 minutos anteriores y la

tercera lectura representa la carga promedio de los 15 minutos anteriores, también se muestra el número de usuarios conectados en ese momento al equipo.

Nagios muestra por separado otra vista en la cual se aprecian sólo los equipos y cuya finalidad es detectar cual de estos esta arriba y detectable. Este vista esta representada por la figura 3.3.

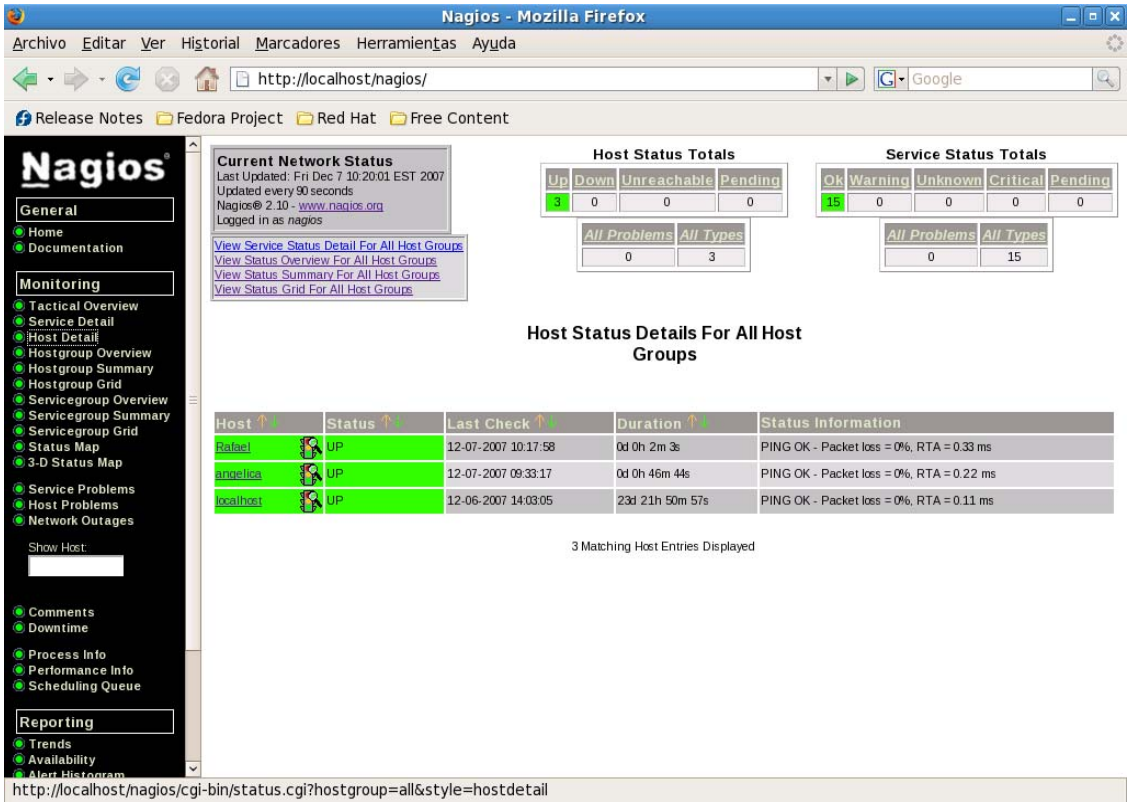


Figura 3.3 Vista de estado por equipo

En esta figura podemos apreciar en el bloque central en la primera columna de izquierda a derecha el nombre del equipo (definido por nosotros), sobre un color verde, el cual nos indica que el servidor fue detectado, la siguiente columna, también de izquierda a derecha, nos indica en texto el estado del servidor, en este caso UP o Arriba (también sobre un fondo verde para acentuar que esta arriba, a su vez si estuviera abajo o inalcanzable, el color de fondo cambiaría a rojo para acentuar ese estado), la tercera columna indica la fecha y hora en la que se realizó la detección, la cuarta columna muestra al igual que la quita columna de la vista de servicios, el tiempo que el equipo ha estado activo, la quinta y última columna de esta vista muestra el que la detección se llevó acabo por medio de PINGs, muestra además el porcentaje de paquetes perdidos y el tiempo promedio de transmisión de éstos.

Nagios permite separar en grupos a los equipos que va a monitorear y como tal también permite ver el concentrado de los grupos de monitoreo. La figura 3.4 muestra los grupos de monitoreo y algunos datos.

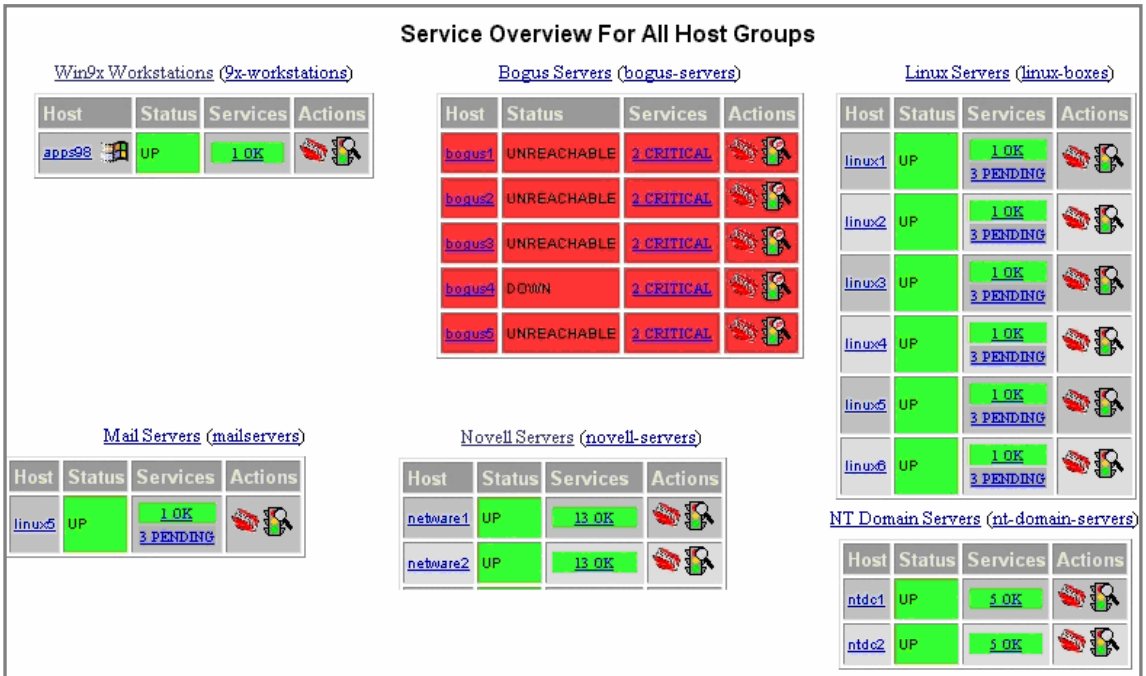


Figura 3.4 Vista preliminar de todos los servicios de todos los grupos de equipos

El concentrado de cada uno de los grupos de equipos proporciona información general del estado de sus miembros.

Nagios también puede graficar los equipos monitoreados en un esquema concéntrico basado en la distancia de Nagios hacia cada uno de los equipos y en el grupo al que el equipo pertenezca. La figura 3.5 muestra esta característica.

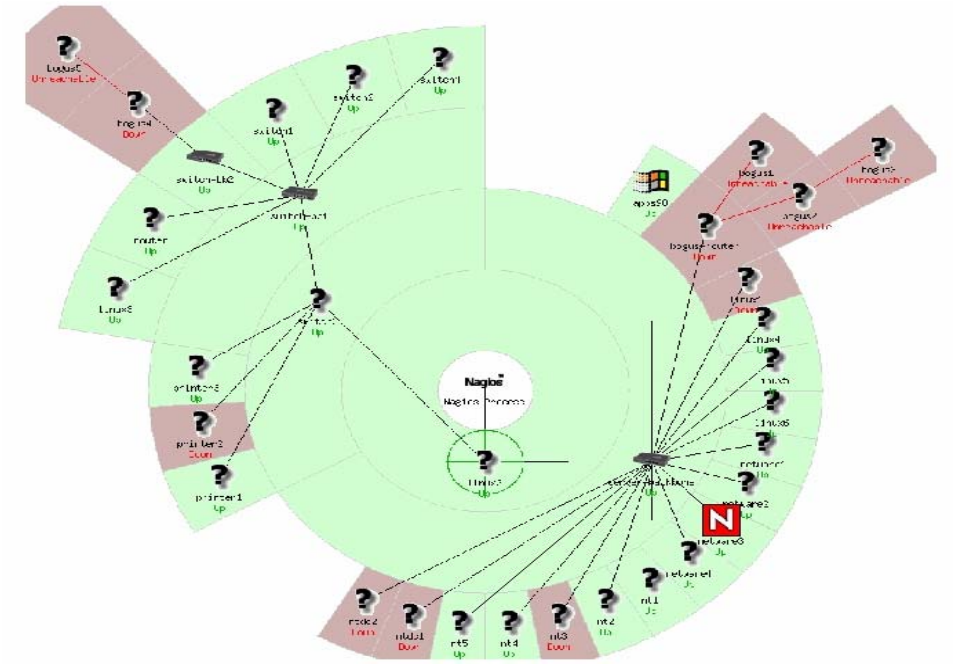


Figura 3.5 Graficación de equipos monitoreados por Nagios

Finalmente Nagios también cuenta con una vista de reportes, en la cual se pueden visualizar tanto las alertas y alarmas, como los eventos exitosos. La figura 3.6 muestra un ejemplo de esta vista.


Figura 3.6 Vista de reportes de Nagios

### 3.3.2 Características de Net-SNMP

Historia

El proyecto Net-SNMP nació en la Universidad de Carnegie-Mellon en 1992. El grupo de redes en CMU (lidereado por Steve Waldbusser) desarrollo una implementación de un relativamente nuevo protocolo de administración de red, el SNMP. La suite programada incluía una biblioteca, una selección comandos SNMP y un agente el cual reportaba la mayoría de la información estándar definida en el RFC 1213. El código fue liberado y puesto a disposición del público.

Davis Wes Hardaker de la Universidad de California extendió el agente para proporcionar más información acerca de su sistema local y para señalar ciertas situaciones de error. Agregó además una manera fácil para la ejecución de scripts y el reporte de resultados. En 1995 este código se liberó públicamente.

A través de los años diversas personas colaboraron en la mejora del proyecto, entre estas, podemos nombrar a Dave Shield de la Universidad de Liverpool, Denmark - Niels Baggesen.

Eventualmente a finales del 2000, el proyecto cambió su nombre a Net-SNMP y cambio su esquema a SourceForge, permitiéndose de esta manera un rango más amplio de respuesta a las tareas de soporte a la administración (en lugar de tener a Wes haciendo todo). Derivado de una serie de reestructuraciones y mejoras en abril de 2002 salió a la luz la versión 5 de Net-SNMP el cual incluía soporte para Windows (gracias a la colaboración de muchos programadores entre los que se pueden mencionar a Mike Slifcak, Robert Story, Alex Burger y Andy Smith).

Al mismo tiempo, el código fue portado a diferentes sistemas operativos, además de ser adoptado como parte de la distribución de la mayoría de los sistemas operativos Linux y BSD's.

Características de Net-SNMP

El SNMP es un protocolo ampliamente utilizado para monitorear la salud o comportamiento de los equipos de red, ruteadores, computadoras o dispositivos como UPS's. Net-SNMP es un conjunto de aplicaciones que implementa los protocolos SNMP v1, SNMP v2 y SNMP v3 utilizando tanto IPv4 como IPv6, la suite de aplicaciones incluye:

- Aplicaciones de línea de comandos para:
  - Recuperar información de un dispositivo con soporte para SNMP, tanto utilizando solicitudes únicas (snmpget, snmp getnext) o múltiples solicitudes (snmp walk, snmptable, snmpdelta).
  - Manipular información de la configuración en un dispositivo con soporte para SNMP (snmpset).
  - Recuperar una colección selecta de información de un dispositivo con soporte SNMP (snmpdf, snmpnetstat, snmpstatus).
  - Convertir de formatos numéricos a contextuales de MIB OIDs, y desplegar el contenido y la estructura del MIB (snmptranslate).
- Un buscador gráfico de MIB (tkmib), utilizando Tk/perl.
- Un demonio de aplicación para recibir notificaciones SNMP (snmptrapd). Notificaciones especificas pueden ser registradas (al syslog, al log de eventos de NT o a un archivo de texto), reenviadas a otro sistema de administración SNMP, o pasado a una aplicación externa.
- Un agente para responder a las consultas SNMP para información de administración (snmpd). Esto incluye soporte interconstruido para un amplio rango de módulos de información, el cual puede ser extendido cargando módulos dinámicos, scripts externos y comandos y tanto el multiplexado de SNMP (SMUX), como los protocolos de agentes de extensibilidad (AgentX).
- Una biblioteca para desarrollar aplicaciones SNMP con APIs de C y Perl.

Dado que Net-SNMP trabaja en base al protocolo SNMP, éste también cuenta con soporte para la versión 3 de SNMP con la cual se puede establecer comunicación cifrada y autenticada. SNMP v3 separa la autenticación y la autorización en dos piezas:

- El USM es el módulo de seguridad por defecto (y el único que Net-SNMP soporta). La U viene de *User-based*, dado que éste contiene una lista de sus usuarios y los atributos de cada uno de ellos. El USM se describe en el RFC 2574.
- El VACM es el Módulo de Control de Acceso basado en Versión y controla cual de los usuarios (y Comunidades SNMP v1/v2c) puede accesar y la manera en que lo hará a las diferentes secciones del árbol MIB. El VACM se describe en el RFC 2575.

Cabe destacar que este no es el comportamiento por defecto de SNMP y por lo tanto tampoco de Net-NMP. Este tipo de comportamiento debe ser configurado previamente tanto en el cliente como en el servidor. SNMP permite autenticar, cifrar o ambos. La autenticación se realiza a través de una llave o firma MD5 o SHA generada a partir de una frase de por lo menos 8 caracteres de longitud. El cifrado de la información se realiza a través de AES o DES.

Cabe resaltar que aunque el esquema de seguridad antes descrito es lo bastante fuerte para desanimar a cualquier novato, no es lo suficientemente fuerte para proporcionar un nivel alto de seguridad, debido principalmente a que tanto la autenticación como el cifrado se basan en aplicaciones consideradas en últimos tiempos vulnerables (con excepción de AES).

### 3.3.3 Características de Cacti

Requisitos

Para la instalación de Cacti es necesario contar con:

- PHP.
- Apache.
- MySQL.
- Net-SNMP.

Características

Cacti es un frontend para RRDTool, el cual almacena toda la información necesaria para crear gráficos y poblarlos con datos en una base de datos MySQL. El frontend esta completamente programado en PHP y posee además soporte para SNMP

Cacti además puede utilizar datos de otros lugares (comandos o scripts externos) en conjunción con cualquier tipo de datos que el usuario defina, para esto sólo es necesario pasarle a cacti las rutas de los datos generados por estos scripts o comandos. Cacti utilizará un trabajo de cron para extraer estos datos y poblar la base de datos de MySQL.

¿Qué es RRDtool?

RRDTool es un acrónimo de Round Robin Database Tool, lo que implica que es una herramienta que trabaja con una Base de Datos que maneja planificación Round Robin. Esta técnica trabaja con una cantidad fija de datos, un proceso y un puntero al elemento actual. La manera en que trabaja una base de datos utilizando Round Robin, es el siguiente; se trata la base de datos como si fuera un círculo, sobrescribiendo los datos almacenados, una vez alcanzada la capacidad de la base de datos. La capacidad de la base de datos depende de la cantidad de información como historial que se quiera conservar.

¿Qué tipo de datos pueden ser almacenados en una RRDB?

Cualquier tipo de dato es admitido, siempre y cuando se trate de una serie temporal de datos. Esto significa que se tiene que poder realizar medidas en algunos puntos de tiempo y proveer esta información a la RRDtool para que la almacene.

Un concepto ligado a las RRDtool es el de SNMP. Este protocolo puede ser usado para realizar consultas a dispositivos acerca del valor de los contadores que ellos tienen (ej: una impresora). El valor obtenido de esos contadores es el que queremos guardar en la RRDTool.

RRDTool, también es ampliamente utilizada en conjunto con otra herramienta llamada MRTG (Multi Router Traffic Grapher), la cual grafica los datos almacenados en la RRDTool. MRTG es una herramienta que grafica datos. Puede ser utilizada para graficar el uso de conexión a Internet, datos como temperatura, velocidad, voltaje, número de impresiones, etc. La RRDTool se utiliza para almacenar y procesar datos recolectados vía SNMP.

En definitiva, para hacer uso de RRDTool, lo que se necesita es un sensor para medir los datos y poder alimentar al RRDtool con esos datos. Entonces, la RRDTool crea una base de datos, almacena los datos en ella, recupera estos datos y los grafica con formato PNG.

Una vez que uno o más fuentes de datos son definidas, los gráficos RRDTool pueden ser creados utilizando esos datos. Cacti nos permite crear casi cualquier tipo de gráfico RRDTool imaginable.

Cacti es una solución completa de red de manera gráfica, diseñada para aprovechar el poder de la funcionalidad de almacenamiento y graficado de RRDTool. Cacti proporciona una extracción de datos rápida, plantillas gráficas avanzadas, múltiples métodos de adquisición de datos y características de administración de usuarios de la aplicación. Todo esto envuelto en una interfase intuitiva y fácil de usar, adecuada para redes LAN y redes complejas con cientos de dispositivos.

Como herramienta de monitoreo, Cacti puede se utilizada junto con Net-SNMP, uno para la presentación de los datos y el otro para la recolección de dichos datos, respectivamente. Un ejemplo de los datos graficados que se muestran en Cacti lo tenemos en la figura 3.7.


Figura 3.7 Gráficas de parámetros de monitoreo contra tiempo

En donde se puede ver la gráfica de la carga promedio, la gráfica de uso de memoria, la gráfica de procesos, la gráfica de usuarios y la latencia de los ping, todos estos datos graficados contra el tiempo.

### 3.3.4 Comparativo

Net-SNMP

Net-SNMP es una herramienta poderosa, aunque no es realmente un sistema de monitoreo, es más bien un conjunto de bibliotecas que implementan al protocolo SNMP en las versiones 1, 2 y 3 de este protocolo, su sintaxis es simple (si se conoce el SNMP), su desempeño rápido y poderoso y su seguridad dependiendo del protocolo a utilizar, puede ser desde nula y altamente peligrosa hasta, segura en un nivel medio.

Es fácil de instalar y su configuración y puesta en marcha requiere de la configuración y puesta en marcha de los componentes de SNMP en los equipos de los cuales se desee obtener información.

Cacti

Cacti tampoco propiamente es una herramienta de monitoreo, más bien es una herramienta pensada para representar gráficamente datos, y dentro del rango de datos que cacti puede presentar de manera gráfica están los obtenidos por Net-SNMP, estos datos pueden ser presentados en gráficas muy vistosas pero en algunos casos poco descriptivas. En general las gráficas presentadas requieren un análisis o conocimiento previo de los datos y de su comportamiento para que realmente signifiquen algo.

Nagios

Nagios si es una herramienta de monitoreo, sin embargo su configuración y puesta en marcha es difícil y algunas cosas que deberían ser configurables desde el entorno Web no lo son, como por ejemplo el alta y configuración del tipo de monitoreo de los equipos. La forma en la que obtiene los datos no es segura. Además muchas de sus vistas son redundantes y su navegación no es ágil.

Las herramientas aquí mencionadas proporcionan ideas interesantes las cuales servirán como base para la implementación de una herramienta que cumpla los requisitos y satisfaga los alcances planteados.

# Capítulo IV

# Sistema de Monitoreo de Servidores Unix

## 4.1 Diseño e implementación del Sistema de Monitoreo de Servidores Unix

Para la creación de la herramienta de monitoreo de servidores Unix, es posible utilizar diferentes metodologías de planificación, todas ellas orientadas a obtener el mejor producto final posible en los tiempos establecidos, sin embargo algunas metodologías se ajustan mejor a unos problemas que a otros. En particular en este proyecto, la metodología que más se apega al tipo de problema planteado es la del modelo incremental, la cual es una combinación de los elementos del modelo lineal secuencial o modelo de cascada y la filosofía interactiva de construcción de prototipos del modelo de construcción de prototipos, en la cual los elementos del modelo lineal secuencial se aplican de manera iterativa en cada ciclo del nuevo prototipo, además en el caso de este modelo a diferencia del modelo de prototipos cada uno de los ciclos genera mejoras o incrementos de funcionalidad con respecto al prototipo anterior, pero siempre respetando del diseño central y siempre con capacidades funcionales desde el primer prototipo.

### 4.1.1 Definición de la problemática

En la actualidad gran cantidad de los procesos productivos, de negocios u operacionales de las organizaciones funcionan o se apoyan en gran medida en sistemas computacionales, los cuales realizan desde tareas muy simples que pueden ser también realizadas por procedimientos manuales o no sistematizados, hasta tareas muy complejas o muy largas que requieren de gran poder de cómputo. La importancia de estos sistemas en las actividades productivas de las organizaciones trae consigo la necesidad de garantizar el buen funcionamiento, la continuidad y la disponibilidad de éstos, motivo por el cual se deben de implementar esquemas orientados a alcanzar estos objetivos.

La revisión frecuente de los dispositivos que proporcionan los recursos necesarios para que las aplicaciones o sistemas funcionen, es una actividad fundamental de la administración de sistemas computacionales, orientada a garantizar la continuidad de los servicios o recursos proporcionados por los sistemas.

La falla, mal funcionamiento, interrupción o indisponibilidad de los sistemas, pueden afectar directa o indirectamente al desempeño de la organización ocasionando pérdidas económicas, pérdida de confianza, pérdida de credibilidad u otras cuestiones nocivas cuantificables o no para la organización.

La información que de los servidores se extrae puede ser valorada de diferentes maneras. En entornos en los que la seguridad es un elemento de importancia, los datos del monitoreo también deben ser protegidos para evitar que sean utilizados por terceros no autorizados que pudieran hacer mal uso de esa información.

### 4.1.2 Análisis de requerimientos del sistema

El monitoreo es un término bien definido y delimitado, sin embargo sus patrones, métricas, objetivos y alcances dependen del tipo de objeto o sistema que se desee monitorear. Existen monitoreos de red, monitoreos de vulnerabilidades, monitoreos de rendimiento y respuesta de aplicaciones, monitoreo de desempeño de bases de datos etc.

El tipo de sistema de monitoreo delineará los parámetros a monitorear, sus posibles valores y significados, los requerimientos, los métodos para extraer la información, el procesamiento que se le dará a los datos y los alcances del propio monitoreo.

En la vida real las tareas de administración del cómputo pueden estar concentradas en un área o en diversas áreas, en un solo administrador o en diferentes administradores, y el alcance de sus privilegios estar o no delimitado. Muchas herramientas de monitoreo no toman en cuenta estas posibles restricciones del entorno, haciendo imposible su utilización en entornos reales en los que el barrido de puertos o el lanzamiento de paquetes tipo broadcast esta restringido a los administradores de la seguridad. Derivado de estas restricciones, surge la necesidad de implementar esquemas de monitoreo que no transgredan las políticas de seguridad ni los ámbitos de ingerencia de las diferentes áreas.

El proyecto de “monitoreo de servidores Unix” nace en un entorno de administración de servidores y como tal, se apega a los privilegios y obligaciones de esta área, el análisis de requerimientos del sistema arrojó los siguientes elementos:

- El sistema estará enfocado a obtener información de forma periódica de servidores Unix y tipo Unix.
- Sólo monitoreará parámetros locales de los servidores.
- La información que de los servidores se obtenga, deberá estar protegida, por tal motivo ésta sólo viajará de manera cifrada.
- Se utilizará un esquema de monitoreo centralizado, con el propósito de evitar la necesidad de un cliente de monitoreo instalado en cada uno de los servidores que se desee monitorear.
- El sistema sólo obtendrá información de los servidores que le hayan otorgado dicho permiso de monitoreo de manera explicita.
- El sistema no realizará pruebas de fortaleza en los servidores.
- Es deseable que el sistema no abra más canales o puertos en los servidores monitoreados y que de ser posible utilice el puerto de administración cifrada estándar.
- La forma de obtener los datos de los servidores deberá ser rápida y sencilla, con el propósito de que el desempeño de éstos no se vea afectado por el proceso de monitoreo.
- Todo el procesamiento de los datos deberá realizase en el servidor central de monitoreo, evitando de esta manera afectar el desempeño de los servidores monitoreados.
- El sistema proporcionará la información del monitoreo a través de una interfase amigable.
- El sistema deberá contar con mecanismos de notificación de eventos.

### 4.1.3 Análisis de requerimientos del software

A pesar de que el monitoreo es una tarea esencial para garantizar la continuidad de la operación de los sistemas computacionales, en general no es una tarea en la cual se desee invertir muchos recursos económicos, más aún en muchos casos los costos de aplicaciones de monitoreo no solo involucran el costo del producto de monitoreo, si no que obligan al consumo otros productos o sistemas operativos para los cuales la licencia es un requisito y que por supuesto implican un costos extra no contemplado.

El análisis de requerimientos de software arrojó los siguientes puntos:

- Es deseable que el costo total del sistema no dependa de licencias de uso de software o de derechos de autor de ninguna clase (bibliotecas, terceras aplicaciones o sistemas operativos).
- El sistema debe se capaz de trabajar en entornos de redes TCP/IP.
- El sistema debe de ser capaz de establecer comunicación cifrada y autenticada con los servidores Unix o tipo Unix.
- El sistema debe de ser capaz de procesar la información extraída en el monitoreo, de manera rápida.

### 4.1.4 Diseño del Sistema de Monitoreo de Servidores Unix

A partir de análisis de requisitos de sistema y del análisis de requisitos de software, se deriva que el sistema debe proporcionar la funcionalidad de monitoreo de manera segura (a través de cifrado y autenticación) y ser desarrollado en un entorno y con herramientas libres para evitar los costos por licencias tanto de software de aplicación como de licencia de sistema operativo.

Derivado de estos requerimientos, se eligió el sistema operativo Linux como entorno para desarrollar y finalmente implementar el sistema de monitoreo de servidores Unix.

Linux es un sistema operativo libre, cuyo ambiente es similar al de los sistemas operativos Unix, el cual además puede establecer comunicación cifrada de manera natural, gracias a que en la mayoría sus distribuciones la aplicación ssh se instala por defecto.

Existen diferentes esquemas para realizar el monitoreo, uno de ello es el monitoreo centralizado (figura 4.1), el cual proporciona ventajas de seguridad con respecto a otros esquemas como el monitoreo basado en cliente/servidor, motivo por lo cual fue seleccionado como esquema de monitoreo para el desarrollo e implementación del sistema de monitoreo. A continuación se presentan sus características.

- Reducción en las tareas y tiempos de administración del monitoreo.
- Reducción de los puntos de contactos y por ende menor exposición de la herramienta (es posible incrementar el nivel de seguridad ya que sólo se tiene que asegurar un punto en la red).
- Detección única de fallas y errores de configuración local y de red.
- Reglas de seguridad de red más simples.
- Reglas únicas de seguridad para los servidores con respecto al servidor de monitoreo.
- La mayor carga de procesamiento se lleva acabo en el servidor de monitoreo, ya que los servidores monitoreados sólo proporcionan los datos requeridos en crudo (sin ningún procesamiento ni pre-formateados), esto permite que el estado real de los servidores monitoreados sea mínimamente afectado por causa del proceso monitoreo.
- Se evita la configuración de clientes en los servidores monitoreados y su posible modificación con el propósito de reportar diferentes valores a los reales.
- La comunicación es unidireccional, es decir, los servidores monitoreados no pueden establecer comunicación hacia el servidor de monitoreo, sólo pueden responder a las peticiones que este le hace.

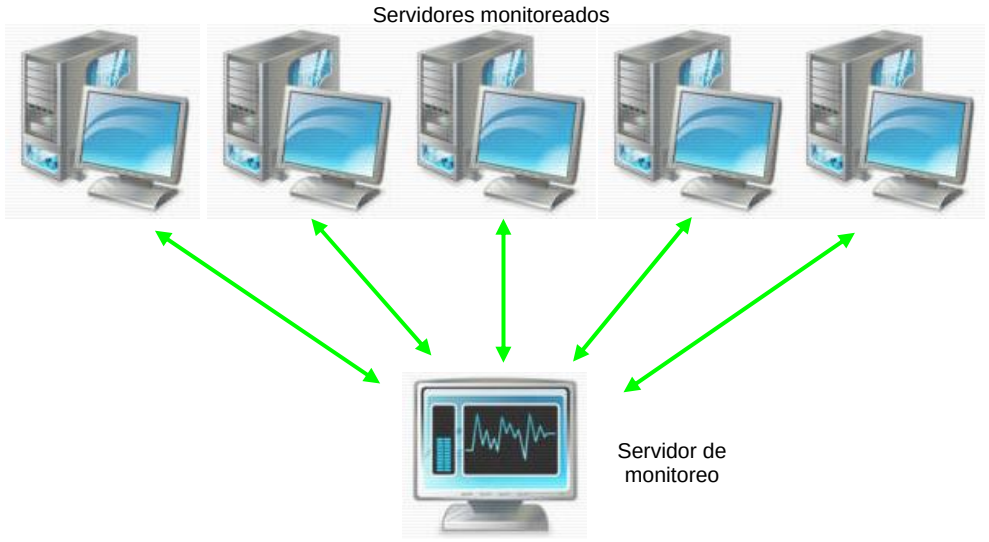


Figura 4.1 Monitoreo Centralizado

Sin embargo, este esquema también cuenta con características en contra, de las cuales la velocidad de respuesta es la más significativa. Esta velocidad o tiempo de respuesta se ve reflejado en los siguientes aspectos:

- En un esquema de monitoreo cliente servidor el cliente lanza un proceso por medio del cual se registra los datos de interés de manera periódica, esto reduce el tiempo de recolección de datos para el servidor de monitoreo, ya que el cliente de monitoreo (en general), entrega los datos recolectados ya formateados.
- El servidor de monitoreo puede recibir una respuesta del cliente a través de la cual se le informe que no hay cambios mayores en el servidor monitoreado y por tal motivo sólo los datos que cambiaron o los que son indispensables en cada ciclo de monitoreo son enviados, reduciendo el tráfico en la red de manera significativa.
- El cliente puede tener características autónomas, de tal forma que cuando se detecte un error o un valor fuera del rango permitido, éste, decida enviar una notificación al servidor incluso sin que este la halla requerido, reduciendo de esta forma el tiempo de respuesta ante una falla.

Como ya se mencionó, la seguridad en la transferencia de los datos de los servidores monitoreados es un requerimiento en el diseño del sistema de monitoreo. Este requerimiento se cubre realizando la comunicación y transferencia de información por medio de la aplicación ssh, el cual como ya vio en el capítulo II, además de cifrar la información puede trabajar bajo un esquema de par de llaves, el cual proporciona autenticación intrínseca al algoritmo asimétrico utilizado, con lo cual se puede garantizar:

- Que las solicitudes para extraer los datos que los servidores monitoreados provienen efectivamente del servidor de monitoreo y no de otra fuente maliciosa.

- Que las solicitudes enviadas van cifradas todo el tiempo durante su viaje a través del canal público no seguro, desde el servidor de monitoreo a los servidores monitoreados.
- Que los datos respuesta de las solicitudes de información, efectivamente provienen sólo de los servidores monitoreados y no de servidores falsos.
- Que los datos recolectados en los servidores monitoreados viajan todo el tiempo cifrados a través del canal público no seguro.

Siguiendo los requerimientos de seguridad, se planteo un diseño de sistema de monitoreo en el cual la parte visible o interfase, interactuara lo menos posible con la parte del sistema encargada de realizar el proceso de monitoreo, con el fin de evitar que a través de la primera se modificase el comportamiento de la segunda de forma no autorizada. Estos componentes se denominaron consola de monitoreo y motor de monitoreo respectivamente y la manera obvia de relacionar a éstos fue a través de una base de datos, la cual estaría colocada de manera lógica en medio del motor de monitoreo y de la consola de monitoreo.

El proceso realizado por el "Sistema de Monitoreo de Servidores Unix" de manera general es el siguiente:

1. El sistema de monitoreo recibe los datos de los servidores a monitorear a través de la interfase de Web (dirección ip, tipo de monitoreo, método de detección y una breve descripción). Estos datos son validados y cargados en la base se datos. La figura 4.2 muestra el proceso antes descrito.


Figura 4.2 Carga de datos de servidores a monitorear

2. Los datos de los servidores son tomados de la base de datos por el motor de monitoreo para identificar los servidores a monitorear, con el propósito de ejecutar los scripts de monitoreo sobre estos últimos. Los datos que se obtienen de los servidores monitoreados son procesados e insertados en la base de datos de monitoreo (figura 4.3).

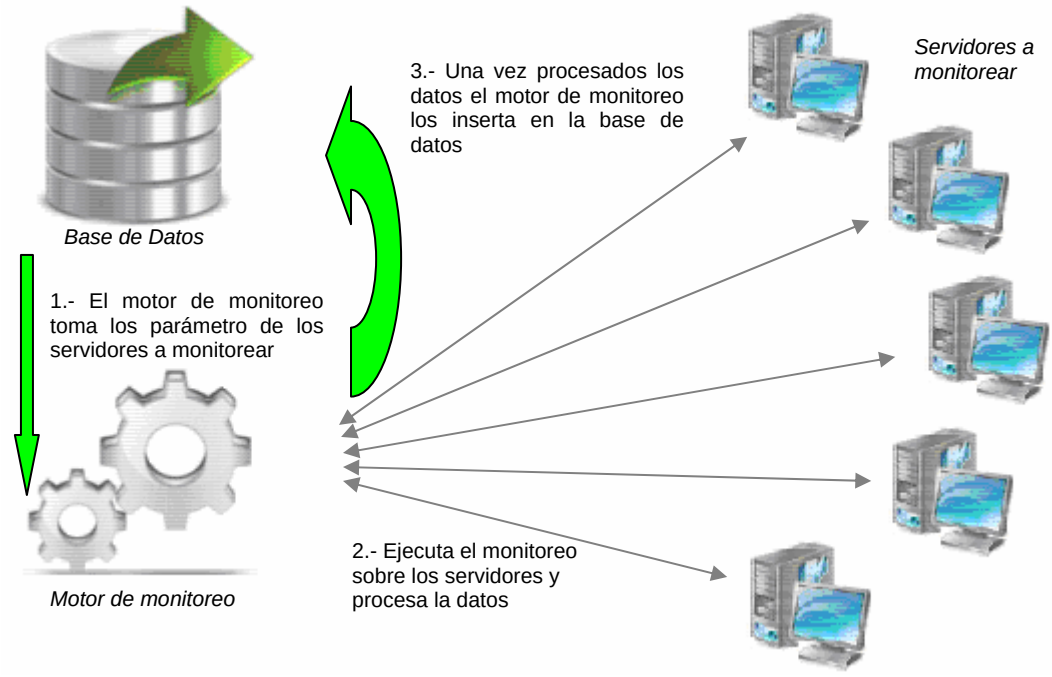


Figura 4.3 Obtención de parámetros, monitoreo y registro de datos de monitoreo

Por último la consola de monitoreo extrae de la base de datos la información obtenida previamente por el motor de monitoreo y en caso de existir alertas las reporta (figura 4.4).

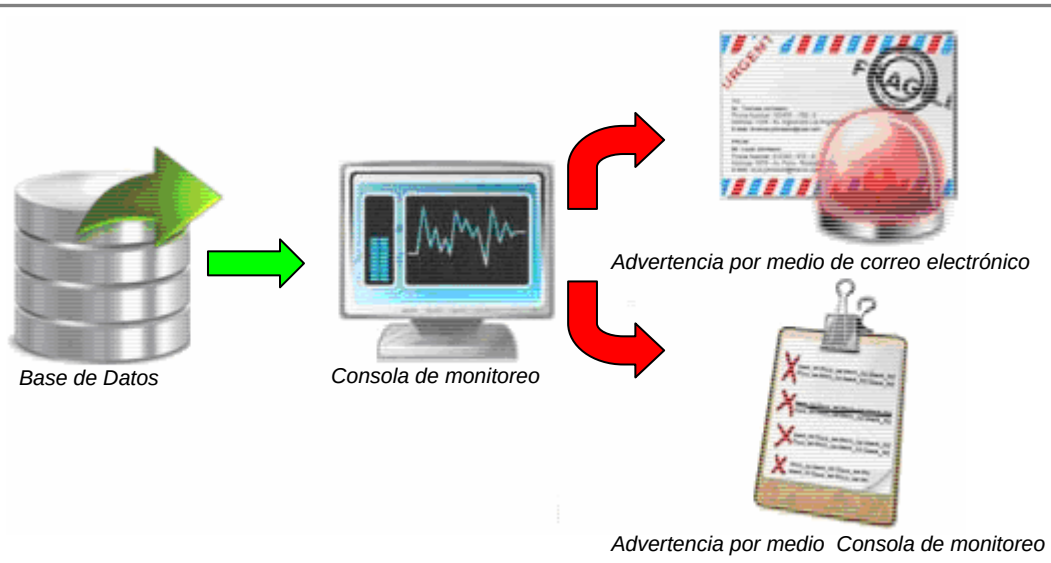


Figura 4.4 Despliegue de información y envío de alertas

A si pues, de esta manera el componente de Web no puede pasar directamente ningún parámetro al motor de monitoreo y todos los parámetros que este último recibe, son pasados a través de un a base de datos. La figura 4.5 muestra la relación entre los tres elementos antes mencionados.


Figura 4.5 Aislamiento del motor de monitoreo

Hasta aquí se ha descrito los requerimientos, esquemas de operación, elementos que componen al sistema y su interrelación general entre ellos. El diseño del sistema de monitoreo de servidores Unix, esta compuesto como ya hemos visto por tres componentes: motor de monitoreo, base de datos y consola de monitoreo. Para la implementación de estos elementos se realizó un proceso de análisis para determinar si las herramientas candidatas eran capaces de proporcionar las salidas esperadas y establecidas por los requerimientos del diseño.

### 4.1.5 Elección del motor de bases de datos

Para el almacenamiento y manejo de los datos extraídos por el motor de monitoreo, se utiliza un motor de bases de datos el cual debe ser capaz de trabajar en entornos Unix y con el lenguaje de programación PHP, para este fin se analizaron dos candidatos de manejadores de bases de datos, MySQL y PostGreSQL. Derivado del análisis de las características de cada una de las dos opciones (descritas previamente en el capitulo II), se seleccionó a MySQL como motor de bases de datos para formar parte la implementación del sistema de monitoreo de servidores Unix (figura 4.6).

Aún cuando si bien, en una primera etapa del desarrollo del sistema de monitoreo el entorno Web no se ha diseñado aún, es necesario contemplar que el motor de bases de datos tenga la capacidad de interactuar de manera eficiente con lenguajes orientados a Web.


Figura 4.6 MySQL como base de datos elegida

### 4.1.6 Herramientas para el motor de monitoreo

El motor de monitoreo es la herramienta a través del cual se establecerá la comunicación del servidor de monitoreo central hacia los servidores monitoreados con el propósito de extraer los datos objetos del monitoreo.

El lenguaje o lenguajes a utilizar deben cumplir con ciertos requisitos básicos, como por ejemplo:

- Soportar el manejo o interrelación con los dispositivos de red.
- Soportar el manejo de comunicación cifrada.
- El lenguaje debe ser compatible con múltiples plataformas.

Para la programación del motor de monitoreo se analizaron diferentes opciones de lenguajes de programación, todos ellos con muchas características y virtudes que los hacían posibles candidatos para la programación del motor.

La lista de lenguajes de programación casi de manera obvia inicio con el lenguaje de programación C, seguido por el lenguaje C++, el lenguaje de programación Perl y por último los interpretes de comandos sh, csh y tcsh, ksh, y bash.

Así pues, finalmente después de analizar cada uno de los lenguajes de programación se definió que el motor de monitoreo sería desarrollado en Bash y Perl (figura 4.7), derivado de las muchas propiedades favorables ya descritas en el capítulo II.

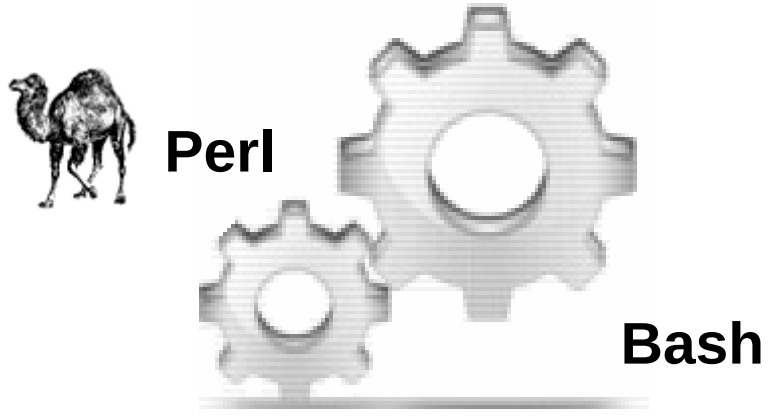


Figura 4.7 Lenguajes del motor de monitoreo

### 4.1.7 Desarrollo del motor de monitoreo

Así como en los casos anteriores, el motor de monitoreo debió circunscribirse a las restricciones de diseño general y a las restricciones de seguridad, dichas restricciones indicaban que el código ejecutado para la extracción de los datos no debía ser modificado. Derivado esta restricción y para asegurar su cumplimiento, se determinó que el código del script no debería transferirse a ninguno de los servidores a monitorear (por lo que tampoco se instalaría un cliente del motor de monitoreo) y que dicho código sólo se ejecutaría a través de la conexión remota.

Para cumplir con el requisito de seguridad, se utilizó el método de par de llaves de ssh, el cual proporciona tres características importantes:

- Autenticación en base al par de llaves del usuario.
- Alto cifrado en la transferencia de datos.
- Permite la ejecución de comandos remotos de manera no interactiva, con lo que se evita la instalación de clientes en los servidores monitoreados.

El procedimiento para la autenticación no interactiva del usuario se lleva acabo de la siguiente manera:

- El usuario genera su par de llaves por medio del comando ssh-keygen (utilizando cualquiera de los algoritmos de cifrado asimétrico disponibles), típicamente rsa. Por ejemplo para generar un par de llaves de algoritmo rsa con una longitud de 2048 se utiliza el siguiente comando:

  ssh-keygen –t rsa –b 2048

  Donde la opción –t indica el tipo de algoritmo a utilizar y la opción –b indica el número de bits a utilizar en el cifrado. Por defecto la longitud utilizada es de 1024 bits, aunque debido a la aparición de mecanismos de factorización especializados y más potentes, se recomienda utilizar 2048 bits.

- Las llaves generadas se almacenan típicamente en el directorio $HOME/.ssh. Para el caso del algoritmo rsa se generarán dos archivos id_rsa e id_rsa.pub, llave privada y llave pública respectivamente (la llave privada NUNCA debe ser compartida, ya que la solidez del algoritmo radica en la improbabilidad de encontrar o derivar una llave privada a través de su correspondiente llave pública).
- La llave pública se comparte con los servidores en los cuales deseamos conectarnos de manera cifrada no interactiva (El contenido del archivo de la llave pública del usuario debe ser copiado de manera íntegra en el servidor a monitorear en un archivo llamado authorized_keys que se encuentra en el mismo directorio .ssh antes mencionado, en este archivo se almacenan la llaves públicas de los servidores que pueden establecer comunicación cifrada no interactiva con el servidor remoto).

Una vez terminado este procedimiento es posible finalmente establecer la comunicación cifrada no interactiva, la cual nos permitirá ejecutar los scripts de monitoreo en los servidores remotos sin la necesidad de autenticarnos por medio de la combinación de usuario y contraseña.

El motor de monitoreo y la base de datos fueron en realidad los primeros elementos generales del sistema de monitoreo de servidores Unix que se implementaron, éstos

se realizaron completamente en un ambiente Linux configurado para realizar conexiones cifradas no interactivas a servidores Unix y Linux, y con MySQL como motor de bases de datos.

Para la primera versión del motor y la base de datos del sistema de monitoreo fue necesario establecer que parámetros de monitoreo eran de interés y relevantes para indicarnos el estado de salud del servidor. Del análisis realizado se determinó que los siguientes elementos eran de interés para el monitoreo de servidores:

- Carga promedio de CPU.
- Frecuencia del Procesador.
- Memoria primaria y secundaria.
- Estado de los sistemas de archivos.
- Actividad en los dispositivos de red.
- Registro de usuarios del servidor.
- Lista de puertos abiertos.
- Lista de dispositivos.
- Datos generales del servidor.

Prototipo 1

En base a estos requerimientos se diseño una primera versión de la base de datos con el propósito de almacenar los datos extraídos por la primera versión del motor de monitoreo, a su vez también se programó un primer motor de monitoreo, el cual tenia como fin establecer los procedimientos básicos para la extracción de la información requerida.

El primer motor de monitoreo fue en realidad un script que contuvo todos los procedimientos necesarios para extraer los datos de monitoreo de los servidores y registrarlos en la base de datos.

En este primer acercamiento todos los datos requeridos fueron extraídos de manera exitosa, pero el tiempo requerido para el término de sus tareas fue demasiado alto, además de que solo registraba los datos en la base de datos y generaba un correo en texto plano para reportar cualquier evento anómalo.

Aún que este primer prototipo cumplía ya con los requerimientos básicos de seguridad y de funcionalidad (al realizar la conexión vía ssh de manera no interactiva a través del mecanismo de par de llaves de usuario y lograr la extracción de todos los datos buscados), su desempeño era demasiado bajo. Este script concentraba todos los códigos necesarios para los datos buscados y ejecutaba el código en un ciclo en el que la lista de control era los datos de los servidores (dirección ip e identificador del servidor), así pues, el código se ejecutaba en cada uno de los servidores y de manera secuencial hasta agotar la lista de éstos.

Este primer prototipo sirvió para determinar los comandos necesarios para la extracción de la información, agrupar datos del mismo tipo y detectar errores y redefinir módulos.

En este prototipo era necesario un promedio de 56 segundos para extraer la información de un servidor, lo que significaba que para una lista de 10 servidores tardaría un promedio de 560 segundos, tiempo extremadamente alto.

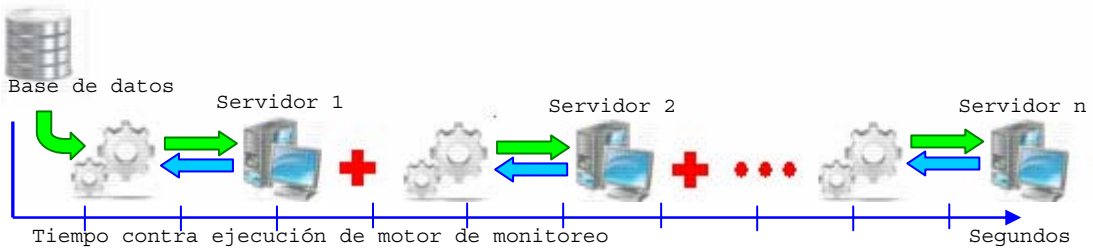


Figura 4.8 Línea de tiempo de la primera versión del motor de monitoreo

La figura 4.8 muestra de manera gráfica el tiempo necesario para realizar el monitoreo en todos los servidores utilizando la primera versión del motor de monitoreo.

Prototipo 2

El segundo acercamiento consistió en un cambio sutil pero efectivo. En lugar de ejecutar el script de manera secuencial esperando a que la ejecución del script terminara para entonces ejecutar el script sobre el siguiente servidor, los datos de los servidores sólo fueron pasados como parámetros al script y la ejecución de éste se realizaría en background y sin esperar al termino de la ejecución del primer monitoreo para iniciar el siguiente y así sucesivamente hasta agotar la lista de servidores.

Esta modificación trajo consigo una importante disminución de tiempo, ya que de manera práctica se lanzaba el script de monitoreo sobre todos los servidores casi al mismo tiempo. Esto redujo el tiempo monitoreo de los servidores a 66 segundos en promedio. Aún así el tiempo de monitoreo era bastante alto, pero en esta ocasión el tiempo promedio dependió más de la programación del script y no del número de servidores involucrado, ya que todos los scripts eran lanzados casi al mismo tiempo.

Este enfoque trajo consigo un problema. Al lanzar todos los scripts casi de manera unísona, no era posible determinar el termino del monitoreo en general, esto debido a que el tiempo de respuesta del script para cada uno de los servidores, dependía de la distancia a la que éstos estuvieran ubicados en la red, y la velocidad con que cada uno de ellos respondiera a las solicitudes, siendo en general estos tiempos diferentes para cada uno de ellos.

Derivado de esta circunstancia se modificó el script para que acusara el término de su monitoreo, a su vez que se programó un script para monitorear el termino del monitoreo general, es decir, el tiempo en el que todos los scripts ejecutados terminaran (figura 4.9).

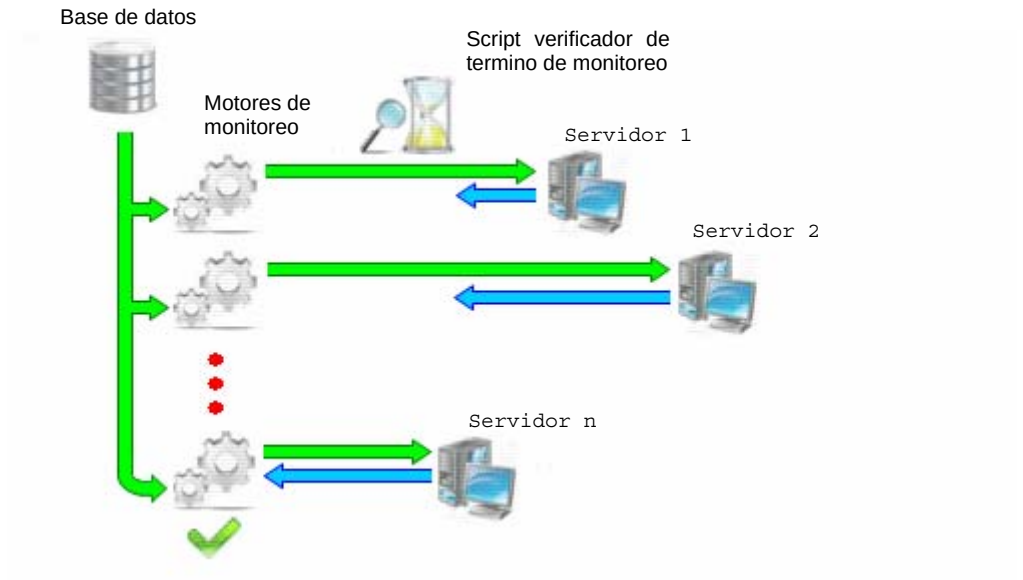


Figura 4.9 Proceso de extracción de datos para la segunda versión del motor de monitoreo

Prototipo 3

Para la siguiente versión se utilizó la filosofía de divide y vencerás, en la que se identificaron elementos de monitoreo similares y se agruparon en módulos independientes, los cuales se programaron en scripts individuales. De esta manera los módulos individuales recibirían los datos de los servidores como parámetros y los ejecutarían en background.

Lo anterior significa que a cada uno de los módulos le son pasados los datos de los servidores y que dichos datos son utilizados como parámetros del script del módulo y posteriormente el script correspondiente a ese módulo y a ese servidor en particular es ejecutado en background, así sucesivamente hasta agotar toda la lista de servidores. Al mismo tiempo este procedimiento se repite para todos y cada uno de los módulos, lo que al final de cuentas resulta en la ejecución de todos los módulos para todos los servidores ejecutándose casi al mismo tiempo (figura 4.10).

Así como en el caso del cambio del prototipo 1 al 2, el cambio del prototipo 2 al 3 representó la creación de nuevos scripts de control para la detección del término de cada uno de los módulos y posteriormente la detección del término del monitoreo, además se crearon módulos independientes para la pre-detección, inicialización y monitoreo de los servidores y un módulo general de arranque y control del motor de monitoreo.

Esta aproximación redujo significativamente el tiempo de monitoreo llevándolo a un promedio de 32 segundos (para pruebas con 10 servidores). Sin embargo el tiempo para el monitoreo de servidores por medio de este prototipo era aún muy alto, por lo que los scripts de los módulos de monitoreo fueron reajustados para realizar un menor número de conexiones por módulo, tanto a los servidores remotos como a la base de datos y de esa manera reducir aún más el tiempo de respuesta.

Otro cambio importante, fue la extracción de datos por medio de scripts de Perl, así como el cambio de código en la mayoría de los ciclos implementados con bash a ciclos implementados con Perl.
Estos últimos reajustes redujeron aún más el tiempo de monitoreo hasta llevarlo a un promedio de 4 segundos, desgraciadamente el tiempo promedio de una conexión de ssh en comparación con el tiempo necesario para extraer y procesar los datos es muy alto.

Por ejemplo, la ejecución de un comando simple como el “ls /etc/” requiere de tiempos promedios muy diferentes cuando se ejecuta de manera local y cuando se ejecuta de manera remota, como puede apreciarse en el siguiente comparativo. Del análisis de estos resultados se puede concluir que el tiempo utilizado en para la extracción y procesamiento de los datos remotos, es casi en su totalidad consumido por la conexión cifrada (aunque este tiempo también depende de la velocidad de la red, en general los resultados no cambian mucho).

Tiempos obtenidos por medio del comando time la ejecutar la línea de comando “ls /etc/”:

| Tiempos locales | |
|---|---|
| real | 0m0.008s |
| user | 0m0.000s |
| sys | 0m0.004s |

| Tiempos remotos | |
|---|---|
| real | 0m0.820s |
| user | 0m0.036s |
| sys | 0m0.004s |

Derivado de lo anterior se concluye que es difícil reducir más el tiempo de monitoreo de manera sustancial. El tiempo utilizado en la conexión de ssh puede disminuirse si se reduce el número de bits en el cifrado de 2048 a 1024, sin embargo, el cifrado recomendado para una conexión de alta seguridad es el 2048 por lo que la longitud de cifrado para requerimientos altos de seguridad no permite negociar está longitud.

Otro elemento importante agregado a este prototipo fue la consola de monitoreo, la cual es la interfase a través de la cual se ingresan los datos de los servidores, se lanzan y detienen los monitoreos, se visualizan los resultados obtenidos en cada una de los ciclos de monitoreo, configuran los valores de ciertos parámetros, se agregan contactos y visualizan alarmas.

La consola de monitoreo es la interfase a través de la cual se visualizan lo datos que se extraen de los servidores monitoreados de manera ordenada y digerida. Interpretan valores de error en los datos e indican de manera gráfica la presencia de dichos errores.

La consola abstrae los datos y los convierte en información fácilmente entendible y que no requiere de interpretación, brindando de esta forma la oportunidad de que usuarios no especializados denominados operadores, trabajen con el sistema de monitoreo sin la necesidad de conocer a detalle el transfondo del monitoreo ni la forma en la que esté se realiza.

La base de datos fue otro de los elementos mejorados en este prototipo, a está, se le agregaron nuevas tablas, se modificaron algunas otras y se eliminaron otras tantas. El diagrama entidad-relación de la correspondiente base de datos se presenta en el apéndice A.

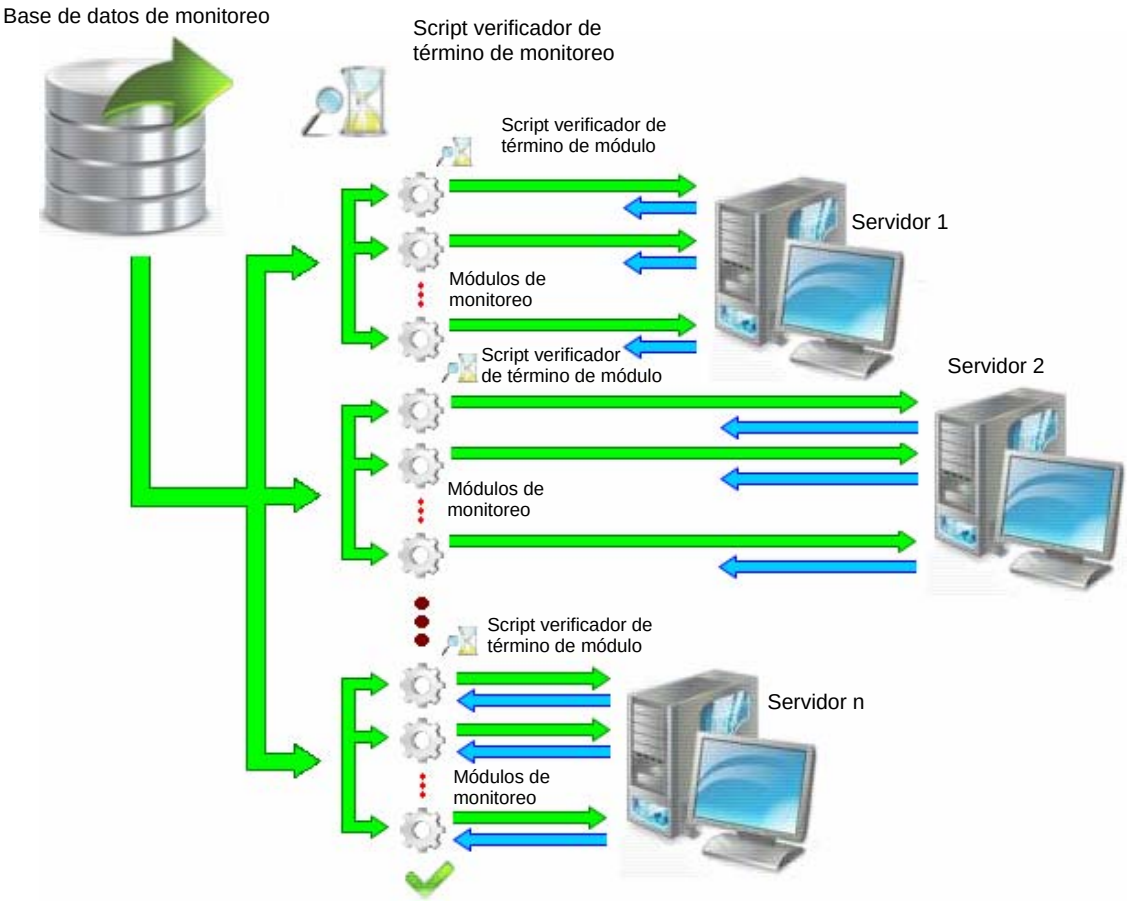


Figura 4.10 Proceso de extracción de datos para la tercera versión del motor de monitoreo

### 4.1.8 Herramientas para la consola de monitoreo

Elección del lenguaje de programación para la consola de monitoreo

La consola de monitoreo es un elemento dinámico por fuerza, dado que la naturaleza propia del monitoreo es el cambio, así pues, la consola de monitoreo deberá proporcionar información actualizada de los servidores monitoreados y permitir la interacción con algunos de los parámetros del sistema.

Derivado de este requerimiento es necesario buscar un lenguaje de programación que nos permita proporcionar dinamismo a la consola de monitoreo, toda vez que el HTML común es estático por definición.

El lenguaje de programación que se elija ha de cumplir con las restricciones de diseño general, por lo tanto esté debe ser un lenguaje capaz de trabajar sobre plataformas Unix o tipo Unix.

Derivado del análisis de las funcionalidades que dichos lenguajes de programación proporcionan (descritas en el capítulo II), se decidió desarrollar la interfase Web o consola de monitoreo con el lenguaje de programación PHP.

Elección del servidor de HTTP

La consola de monitoreo debe de ejecutarse sobre un servidor de http y al igual que el lenguaje de programación, éste debe de circunscribirse a las restricciones definidas por el diseño general.

Existen muchos servidores de http, algunos comerciales, otros libres (AOLServerm Roxenm Thttpd, httpd Sun, Lighttpd, ISS etc.), pero sin duda Apache es el servidor líder en Web (por lo menos a si lo muestran las estadísticas de servidores activos en la red hasta diciembre de 2007), con el 50.76% del total de dominios activos en el Internet, seguido por su más cercano perseguidor el servidor IIS con 35.84% del total de servidores web en la red (142,805,398 de servidores de Web activos que reportan tener instalado Apache hasta diciembre de 2007).

Se decidió que la consola de monitoreo estuviera desarrollada en un entorno web derivado de las bondades que este entorno proporciona (fácil acceso, esquema que permite la seguridad y alta disponibilidad). También se decidió que el servidor de http fuera Apache y los lenguajes de programación principales PHP, HTML y CSS, derivado a sus muchas virtudes (descritas ya en el capitulo II).

Finalmente, como se muestra en la figura 4.11, la consola de monitoreo se compone de un servidor Apache y PHP, HTML y CSS como lenguajes principales de programación.


Figura 4.11 Componentes principales de consola de monitoreo

### 4.1.9 Diseño de la consola de monitoreo

La función de la consola de monitoreo es presentar la información de manera oportuna, concisa y amigable, de forma tal que no sea necesario que la persona que visualiza la información tenga el conocimiento técnico ni el detalle de la forma en la que se extraen los datos, permitiendo de esta manera que no sólo el administrador de los sistemas pueda operar la aplicación, sino también personas menos especializadas,

por ejemplo un operador, al cual se le pude asignar la tarea de vigilar la consola en espera de eventos, permitiendo de esta manera que el administrador se enfoque en actividades más sustanciales.

A través de la consola de monitoreo es posible definir los servidores que se desean monitorear, el tipo de monitoreo que se desea aplicar a éstos (monitoreo total o sólo detección), la frecuencia de monitoreo, los valores límite de algunos parámetros, definir puertos, agregar contactos para las alarmas, arrancar y parar el motor de monitoreo (tanto de manera general como de manera individual) y agregar usuarios con diferentes perfiles.

La consola, como componente del sistema de monitoreo, esta también acotada por las restricciones de diseño, en las cuales la seguridad es un punto importante, de tal forma que el diseño de este componente también contempla cuestiones de seguridad desde su diseño particular. Del análisis de requerimientos del sistema se determinó que la consola de monitoreo debe implementar los siguientes elementos:

- Un método control de acceso.
- Un método para la transferencia segura (cifrada y autenticada) de los datos presentados en la consola de monitoreo.
- Un esquema de diferentes perfiles de usuarios (permisos o privilegios diferentes para cuestiones administrativas).
- Un método para la restricción de ejecución de código arbitrario.
- Un método para evitar consultas no permitidas a la base de datos.
- Presentar los datos de manera concisa y en formatos simples.
- Utilizar métodos basados en colores para identificar las alertas de manera fácil y rápida.

Del análisis de requerimientos del software se desprendió que el método más práctico para proporcionar el acceso al sistema de manera segura, era la combinación de nombre de cuenta de usuario y contraseña, el cual, a través de un esquema de sesión proporcionaría al sistema el valor asociado a los privilegios del usuario durante todo su tiempo de navegación en el sistema. Dicho de otra forma sin la combinación correcta de nombre de cuenta y contraseña no se genera la cadena de sesión, la cual es indispensable para la navegación a través las páginas que componen el sistema.

Se generó un diseño básico, en el cual a través de una página acceso se realizaría el registro del nombre de usuario y contraseña, una vez validado el usuario, el sistema desplegaría la página principal y a través de ella se realizaría la navegación en el sistema auxiliándose de los diferentes menús y botones.

La seguridad se concentró en la página inicial del sistema (index.php), en la cual se colocó el módulo de autenticación, y la única conexión a la base de datos, de tal manera que los archivos contenidos en la estructura de directorios del sistema no ejecutan ninguna de las consultas en ellos programadas aún cuando sean invocados de manera directa, garantizando así que la única forma de interactuar con la base de datos sea a través de la página antes mencionada (siempre y cuando el resultado de la autenticación halla sido satisfactorio, figura 4.12).


Página de acceso

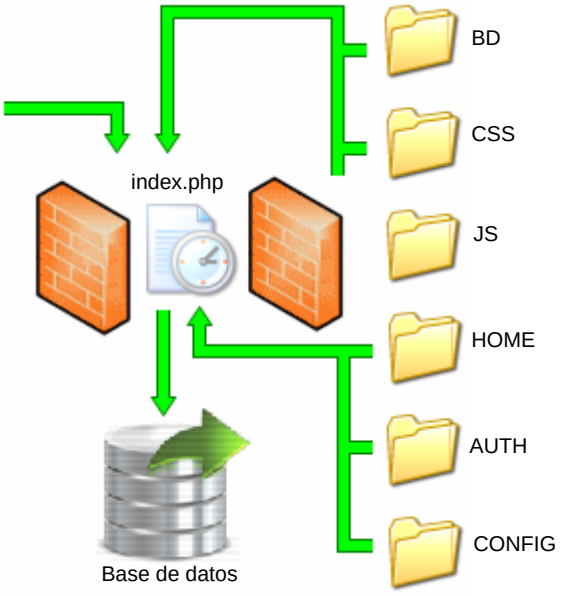


Figura 4.12 Diseño de la consola de monitoreo

En la figura 4.12 además se puede apreciar la estructura de directorios propuesta para la consola de monitoreo, la cual se describe a continuación:

| | |
|---|---|
| DB: | Directorio para los archivos de consultas a la base de datos. |
| CSS: | Directorio de archivos de hojas de estilo. |
| JS: | Directorio de archivos de funciones javascript. |
| HOME: | Directorio de archivos de plantilla y de funciones de php. |
| AUTH: | Directorio de archivos para la implementación de la autenticación. |
| CONFIG: | Directorio de archivos de configuración de la consola de monitoreo. |

Para evitar que los datos que se envíen y reciban a través de la consola de monitoreo puedan ser visualizados por terceros no autorizados, es necesario que dichos datos sean cifrados antes de viajar a través del canal público no seguro. Este cifrado puede ser implementado en Apache por medio del módulo mod_ssl, el cual brinda una interfase a la biblioteca OpenSSL, la cual proporciona alto cifrado utilizando los protocolos Secure Sockects Layer y Transport Security Layer.

A partir de los requerimientos de diseño, se definió que la consola de monitoreo presentará los datos en una sola página en la cual solo fuera necesario seleccionar los diferentes botones para navegar entre las diferentes opciones de información. Así mismo se seleccionaron colores suaves en tonos azules para proporcionar la sensación de seguridad y colores rojos intensos para indicar errores o alarmas.

### 4.1.10 Pruebas

Las pruebas realizadas al sistema, se dividieron en dos partes las pruebas de despliegue de información en la consola de monitoreo y las pruebas de funcionalidad en el motor de monitoreo, dado que el número de datos obtenido de los servidores era relativamente pequeño, la respuesta de la base de datos hacia la interfase Web fue casi inmediata en todas sus pantallas y los datos presentados fueron correctos y

acordes a lo obtenido por el motor de monitoreo. Por otro lado, es realmente el motor de monitoreo el que se encarga de conectarse a los servidores, extraer los datos, analizarlos, darles formato, catalogarlos y distribuirlos en las diferentes tablas de la base de datos, así como de realizar la semaforización de sus procesos y la verificación de termino de todos los módulos en todos de todos los servidores.

Para la implementación de las pruebas no fue necesario desarrollar ningún programa adicional, dado que el motor de monitoreo contaba ya con un script para detectar el termino total del monitoreo, por lo que fue relativamente fácil medir el desempeño del motor de monitoreo, es decir, solo bastó con registrar el instante de inicio del monitoreo y el instante de termino de todos monitoreos en los servidores.

Para registrar estos tiempos simplemente se agrego el comando “date” con el símbolo de & (para que su ejecución se realizase en background), para que de este modo su salida de datos no agregase tiempo al proceso de monitoreo, de igual manera se agrego el comando “date” al final del monitoreo.

Al realizar las pruebas en cuatro máquinas del segmento 132.248.54.X de la Facultad de Ingeniería en el Departamento de Redes y Operación de Servidores – UNICA, se obtuvieron los siguientes resultados, los cuales fueron plasmados en la siguiente tabla:

| Rango de lecturas | Lecturas en formato hh:mm:ss | | | | |
|---|---|---|---|---|---|
| 1 al 5 | inicio 12:45:08<br>fin 12:45:010 | inicio 12:45:23<br>fin 12:45:25 | inicio 12:45:39<br>fin 12:45:40 | inicio 12:45:53<br>fin 12:45:56 | inicio 12:46:08<br>fin 12:46:10 |
| 6 al 10 | inicio 12:46:23<br>fin 12:46:25 | inicio 12:46:38<br>fin 12:46:40 | inicio 12:46:53<br>fin 12:46:55 | inicio 12:47:08<br>fin 12:47:11 | inicio 12:47:23<br>fin 12:47:26 |

Tabla de duración de monitoreos

Del análisis de los datos en la tabla de duración de monitoreos se desprende que el tiempo promedio del monitoreo para estos cuatro servidores fue de 2.2 segundos, lo cual es un tiempo suficientemente corto para una herramienta de monitoreo.

Sin embargo es de esperarse que cuando número de equipos en monitoreo oscile entre 30 y 50, el tiempo de monitoreo se dispare, ya que cada una de las conexiones estará compitiendo por acceder al medio.

## 4.2 Implementación del Sistema de Monitoreo de Servidores Unix

El “Sistema de Monitoreo de Servidores Unix” es un sistema que nace a raíz de necesidades particulares de seguridad. La información que de él se puede extraer, se presenta de manera rápida, directa y en formatos claros, sin embargo el sistema de monitoreo sólo cubre en esta etapa los parámetros básicos necesarios para determinar de manera efectiva del estado de salud de los servidores. El sistema esta diseñado para que más módulos sean agregados, incrementando de esta manera su funcionalidad.

La amplitud y diferencia entre los diferentes sistemas operativos tipo Unix, pero sobre todo entre los sistemas operativos Linux hace difícil abarcar con exactitud a todas las distribuciones, más sin embargo no imposible.

Los objetivos del proyecto han sido alcanzados, dado que se implementó una herramienta capaz de proporcionar niveles de seguridad altos en el proceso de monitoreo de servidores extrayendo información valiosa e importante para la toma de decisiones.

Cuando se habla de monitoreo de sistemas computacionales, por fuerza hay que hablar valores base o de línea base, dichos valores nos indican, definen o están asociados a estados de desempeño de un parámetro en particular, los valores de línea base son la cota inferior en un análisis de desempeño, es decir, es el estado para el cual ese parámetro tiene un comportamiento óptimo, en contraposición con los valores críticos o valores máximos que son los valores para los cuales los parámetros presentan comportamientos erráticos. Dichos valores de línea base son observados para el desarrollo de la herramienta de monitoreo.

A continuación se presenta una tabla en la cual se presentan valores de línea base y críticos para algunos de los parámetros de monitoreo.

| Parámetros | Descripción del parámetro | Línea base | Valores críticos |
|---|---|---|---|
| Procesador | El CPU es uno de los componentes más rápidos de la computadora. Si el comportamiento del CPU alcanza 80% ó 90% de su utilización, el desempeño general del sistema se verá afectado. | 10% a 15% en servidores con sistemas Web de mediana y alta demanda y motores de bases de datos pequeños. | 80% ó 90% |
| RAM | La memoria física primaria es un elemento indispensable para la ejecución de los procesos, sin ella los tiempos de procesamiento serian increíblemente mayores a los actuales. | El valor de línea de base debe ser tomado respecto a cada uno de los servidores en el momento de iniciada la operación (con aplicaciones y servicios ejecutándose tal cual se ejecutarán cuando inicie la fase de producción). | 80% ó 90% |
| SWAP | La Swap o memoria virtual es una técnica que proporciona la impresión a una aplicación o proceso de que éste tiene siempre la suficiente memoria para trabajar, cuando de hecho muchas veces es necesario almacenar los datos del proceso o aplicación en espacios especiales del disco duro, en espera de que la memoria física o RAM se desocupe para dar a alojamiento al proceso o sus datos. Los procesos que utilizan la Swap se alentan, ya que la escritura y recuperación los datos de disco es mucho más lenta en comparación con la escritura y la lectura en las memorias físicas. | El comportamiento de la memoria Swap, difiere entre los sistemas operativos Unix y los Linux, mientras que en algunos Unix el tamaño de la memoria Swap debe ser por lo menos del mismo tamaño que la memoria RAM como requisito para poder hacer uso de esta última y más grande si se requiere ejecutar procesos que sobre pasen el tamaño de la memoria primaria ó RAM, en el caso de los sistemas operativos Linux, hace ya algunos cuantos años la memoria swap no es indispensable, siempre y cuando la memoria RAM sea lo suficientemente grande para alojar a los proceso en turno. Más sin embargo, la Swap no es mala idea, ya que permite contener procesos que sobre pasen la capacidad de la RAM para manejarlos. La utilización de la memoria Swap trae consigo en todos los casos, un decremento en el desempeño y en general se toma como regla que su uso sea menor al 1/3 del total de ella para valores de línea base. | mayor al 66 % |

Tabla de valores de línea base

| Parámetros | Descripción del parámetro | Línea base | Valores críticos |
|---|---|---|---|
| Sistemas de Archivos | Un sistema de archivos en Unix o Linux es una estructura arborecente diseñada para almacenar, controlar y administrar archivos de diferentes tipos y directorios. | El valor de línea de base para un sistema de archivos en particular, no se define en porcetaje y depende del tipo de uso que se le de. Por ejemplo, un sistema de archivos asignado para almacenar las aplicaciones, bibliotecas y archivos binarios de un sistema operativo, no tendrá el mismo comportamiento que el sistema de archivos utilizado para el manejo y almacenamiento de los datos de una base de datos. En el primer caso, es poco probable que el espacio utilizado para almacenar las bibliotecas, binarios y demás archivos crezca demasiado una vez estabilizado el servdor con respecto a los recursos que utilizarán en condiciones normales de operación, en el segundo caso, el tamaño del sistema de archivos debe ser lo sufiencientemente grande como para soportar el crecimiento dinámico de datos, propio de las bases de datos. Más aún, un valor de 60% de utilización no significa que el sistema de archivos tenga espacio sufiente para continuar su operación, ya que el sistemas de archivos podría ser de 500 MB y crecer a un ritmo de 50 MB por día, en cuyo caso el espacio disponible se acabaria en pocos dí | No definido / MB o GB |
| Puertos de Red | Los puertos de red son puertas lógicas que se abren en la intefase de red y que permiten establecer comunicación de manera ordenada y especifica a los protocolos y aplicaciones. El valor de los puertos en redes TCP/IP v4 proporciona 35535 opciones o puertos. | El estado de línea base para los puertos de red, se define como los puertos necesarios e indispensables para llevar acabo las tareas que requieran conexión a red. Debido a que los puertos representan una posible entrada al sistema de manera es nesario solo utilizar aquellos puertos cuyas tareas así lo justifiquen. | Solo los puertos necesarios abiertos de manera autorizada |

Tabla de valores de línea base

El “Sistema de Monitoreo de Servidores Unix” es relativamente fácil de instalar, pero requiere de permisos de administrador para algunas cuestiones de configuración. A continuación se presentan los pasos necesarios para poner en operación el sistema de monitoreo.

### 4.2.1 Requisitos de Instalación

La instalación del “Sistema de monitoreo de servidores Unix” requiere en realidad muy poco para su instalación, básicamente el sistema puede ser instalado en cualquier computadora que cuente con tarjeta de red y que soporte un sistema operativo Linux, debido a que los recursos (procesador y memoria RAM) que demanda son pocos.

Los elementos de la siguiente lista se presentan como requisitos, pero en realidad el sistema debería trabajar bien bajo casi cualquier entorno Linux debido a que la programación es básica y muy general, además de que no se compila nada, así que no existen dependencias con la plataforma.

- Sistema operativo Unix ó tipo Unix (Linux Fedora Core 5).
- Ssh u openssh 4.7p1 ó superior.
- MySQL 5.0 ó superior.
- Apache 2.2.X ó superior con PHP 4.2.X ó superior.
- Bash y Perl.

Tanto ssh, bash y perl son parte por defecto de la mayoría de las distribuciones de Linux. Los requisitos de versiones, son más una cuestión de seguridad que de funcionalidad derivados de la importancia en la seguridad del servidor donde residirá el sistema de monitoreo.

### 4.2.2 Configuración inicial e instalación

El sistema de monitoreo de servidores Unix se distribuye en un archivo tipo tar.gz y una vez descomprimido genera un directorio llamado SIMSU, el cual contiene cuatro directorios: docs, monitoreo, motor, sql y dos archivos: LEEME.txt y simsu (figura 4.12).

```
[root@tepocata hackme]# ls SIMSU.tar.gz
SIMSU.tar.gz
[root@tepocata hackme]# tar -zxf SIMSU.tar.gz
[root@tepocata hackme]# cd SIMSU
[root@tepocata SIMSU]# ls
docs  monitoreo  motor  sql  LEEME.txt  simsu
```

Figura 4.12 Paquete SIMSU

En el directorio “docs” se localiza la documentación en formato html, el directorio “monitoreo” contiene la aplicación de Web o consola de monitoreo (este directorio debe ser colocado en alguna ubicación por debajo del directorio raíz de http), el directorio “motor” como su nombre lo indica, contiene al motor monitoreo (este directorio no debe estar dentro del directorio raíz del servidor de http), el directorio “sql” contiene una copia de la base de datos y dentro de este directorio también se generará un script para instalar la base de datos con el usuario definido en la configuración, el archivo “LEEME.txt” es una breve descripción de la estructura de directorios de sistema de monitoreo y los pasos rápidos para su instalación, por último el archivo “simsu” es un instalador que deberá ejecutarse previa configuración del archivo “motor/conf/monitoreo.conf”.

El directorio motor se compone de seis directorios: archivos_de_control, conf, demonios, logs_de_error, scripts, temporales y un script llamado montoreo_servidores.sh con el cual se levanta el motor de monitoreo, (figura 4.13).

```
[root@tepocata motor]# ls
archivos_de_control  conf  demonios  logs_de_error
monitoreo_servidores.sh  scripts  temporales
[root@tepocata motor]#
```

Figura 4.13 Contenido del directorio motor

La descripción general del contenido de los directorios se presenta a continuación:

- archivos_de_control
  - Contiene archivos tipo PID, es decir de números de procesos.
- conf
  - Contiene archivos de configuración del motor de monitoreo.
- demonios
  - Contiene scripts de las funciones generales de monitoreo (detección, inicialización y monitoreo).

- logs_de_error - Contiene archivos de error que pudieran generarse durante la ejecución de cualquiera de los procesos de monitoreo.
- scripts - Este directorio contiene los scripts de los diferentes módulos de monitoreo.
- temporales - Este directorio contiene archivos temporales generados durante la ejecución del procesos de monitoreo.
- monitoreo_servidores.sh - Script central que controla las solicitudes de monitoreo.

El directorio “conf” contiene un archivo llamado “monitoreo.conf” el cual debe ser modificado para que el sistema de monitoreo trabaje de acuerdo a los parámetros particulares de cada servidor. El archivo “monitoreo.conf” tiene el formato presentado en la figura 4.14.

```
[root@tepocata SIMSU]# more motor/conf/monitoreo.conf
BD_HOST="localhost"
BD_NOMBRE="monitoreo"
BD_USUARIO="monitoreo"
BD_PASSWORD="sepersecreta_1"
RUTA_SERVIDOR_HTTP="/usr/local/apache2/htdocs/"
DIRECTORIO_APLICACION_WEB="mon_mon"
BIN_DIR_NMAP="/usr/local/bin"
USUARIO_REMOTO="hackme"
LLAVE_PRIVADA="/home/hackme/.ssh/id_rsa"
[root@tepocata SIMSU]#
```

Figura 4.14 Archivo de configuración monitoreo.conf

En donde:

- BD_HOST - Es la máquina que contiene la base de datos.
- BD_NOMBRE - Es el nombre de la base de datos (por defecto monitoreo).
- BD_USUARIO - Usuario administrador de la base de datos (por defecto monitoreo).
- BD_PASSWORD - Contraseña de la base de datos monitoreo para el usuario monitoreo.
- RUTA_SERVIDOR_HTTP - Ruta del directorio raíz de publicación de contenido del servidor http.
- DIRECTORIO_APLICACIONES_WEB - Nombre o ruta relativa del directorio de la consola de monitoreo dentro del directorio de publicación de http.

Una vez definidos los parámetros del archivo de configuración (monitoreo.conf), se ejecuta el script de instalación simsu, el cual realizará la validación tanto de los pre-requisitos de instalación, como del los datos del archivo de configuración y posteriormente instalará el componente Web ya parametrizado (figura 4.15).

```
[root@tepocata SIMSU]# ./simsu intalar

------------------  Verificando Requisitos  ----------------------

Binario de NMAP detectado                                    [OK]
Binario de HTTPD detectado                                   [OK]
Binario de MySQL detectado                                   [OK]
Motor de SMTP detectado en el puerto 25                      [OK]


------------------  Verificando configuración  -------------------

Host de la base de datos "localhost"
.....................................................        [OK]
Nombre de la base de datos "BD_test_1"
.....................................................        [OK]
Nombre del usuario base de datos "user_test_1"
.....................................................        [OK]
Usuario remoto "hackme"
.....................................................        [OK]
Directorio de http:  /usr/local/apache2/htdocs/
.....................................................        [OK]
Directorio de aplicación:  mon_mon
.....................................................        [OK]
Ruta de NMAP /usr/local/bin/nmap
.....................................................        [OK]
Nombre de cuenta del usuario remoto hackme
.....................................................        [OK]
Ruta de llave privada /home/hackme/.ssh/id_rsa
.....................................................        [OK]


------------------  Preparandose para Instalar  ------------------
Copiando archivos y permisos a la ruta de HTTP
/usr/local/apache2/htdocs/mon_mon
.....................................................        [OK]

Ejecute el siguiente comando con un usuario MySQL con
privilegios para crear bases de datos y usuarios:

mysql -u usuario_privilegiado -p < sql/genera_bd.sql
```

Figura 4.15 Ejecución del instalador del Sistema de Monitoreo de Servidores Unix

Una vez ejecutado el instalador se generará una copia del directorio “monitoreo” en la ruta y con el nombre definido en el archivo de configuración. Suponiendo que el nombre del directorio no se modifico, éste se llamará “monitoreo” y contendrá los subdirectorios mostrados en la figura 4.16.

```
[root@tepocata ~]# cd /usr/local/apache2/htdocs/
[root@tepocata htdocs]# ls
monitoreo
[root@tepocata htdocs]# cd
[root@tepocata monitoreo ]# ls
acceso.php  autentica  cal  css  home
img  index.php  js  temporales
[root@tepocata monitoreo]#
```

Figura 4.16 Contenido directorio monitoreo

El directorio monitoreo se compone de los siguientes subdirectorios y archivos:

- autentica - Código para la autenticación y control de las sesiones de usuario.
- cal - Código del calendario
- css - Hojas de estilo.
- home - Código de la consola de monitoreo
- img - Imágenes de la consola de monitoreo
- js - Código javascript
- temporales - Archivos temporales
- acceso.php - Página de registro a la consola de monitoreo
- index.php - Página de inicio de la consola de monitoreo

En el directorio “autentica” existe un archivo llamado “aut_config.inc.php”, el cual contiene los parámetros de conexión a la base de datos, y el nombre del directorio base de la parte de la aplicación para Web, este archivo se genera automáticamente y toma sus valores del archivo de configuración “motor/conf/monitoreo.conf” (figura 4.17).

```
<?php
                                        // Datos conexión a la Base de datos (MySql)
$sql_host="localhost";                  // Host, nombre del servidor o IP del servidor Mysql.
$sql_usuario="user_test_1";             // Usuario de Mysql
$sql_pass="sepersecreta_1";             // contraseña de Mysql
$sql_db="BD_test_1";                    // Base de datos que se usará.
$sql_tabla="Usuarios";                  // Nombre de la tabla que contendrá los datos de los usuarios

$usuarios_sesion="AUTENTICACION_DE_APLICACION";

$DIRECTORIO_BASE="mon_mon";             // Nombre del directorio de interfase
?>
```

Figura 4.17 Archivo de configuración para el componente de Web

El siguiente paso es compartir la llave pública del usuario a los servidores a monitorear (figura 4.18). Este procedimiento permitirá que el servidor de monitoreo se comunique a los servidores monitoreados sin la necesidad de registrarse de manera interactiva en los servidores.

```
[root@tepocata ~]# cat /home/hackme/.ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA34xiJesE9oqH4jleEx9wy3dXJCLYcJ5+Jjjmjx34V6yj
6vmxzfe/+dXK4zypQ72+avdOknbxVyHkLZOpbFjUhsv9XgwI8F1ZOkrd5QkOyL1wMcS3xOhPzOLnn9SS
+gcn/XNDhUhCdFTqMOV8AlAun3ZhKTaCvmV2pmrLMO2ogrdmObOXY5DxTmcdvEmFlr7KUiWhtg/z7T88
PWlhBmBStYfEyvOjCRmSuBOTrPGs6Pl9O/wcXYHsdJ17LtKHhDWutkQHGC1Yq5lNfimVqyLRVgTCkEcR
uZB/hXx4aqyKbVwfbkUAQvnPAmYzQn8BGUIo7FowfEWmk2IFdLZ92GhBuw== hackme@tepocata.intranet
[root@tepocata ~]#
```

Figura 4.18 Llave pública rsa

Esta llave se guarda en los archivos authorized_keys del directorio .ssh del usuario y servidor correspondiente. Para la comunicación entre servidores y la generación de la llave se utiliza un usuario no privilegiado, el cual debe también existir en los servidores a los que se conectará.

### 4.2.3 Navegación y funcionalidad del sistema

El sistema de monitoreo requiere la autenticación del usuario para accesar a él, más sin embargo esta autenticación sólo es un método para permitir el acceso al sistema y no para proteger los datos, es decir, tanto los usuarios como las contraseñas viajan en claro, así como todos los datos del monitoreo, por lo que la preparación del servidor de http con cifrado es un requisito para la implementación del sistema de monitoreo de servidores Unix.

La configuración de apache con ssl no es el tema de este documento, por lo tanto se dará por sentado que esta configuración no representa problema alguno.

Para ingresar al sistema de monitoreo es necesario abrir un navegador de Internet y teclear la siguiente dirección URL: https://nombreoipservidor/monitoreo/acceso.php, donde “nombreoipservidor” corresponde al nombre o dirección ip del servidor en donde se instaló y configuró el sistema de monitoreo de servidores Unix. En esta página se ingresará el nombre de la cuenta del usuario en la caja de texto “Usuario” y la contraseña correspondiente a esa cuenta, en la caja de texto “Contraseña” como se muestra en la figura 4.19.

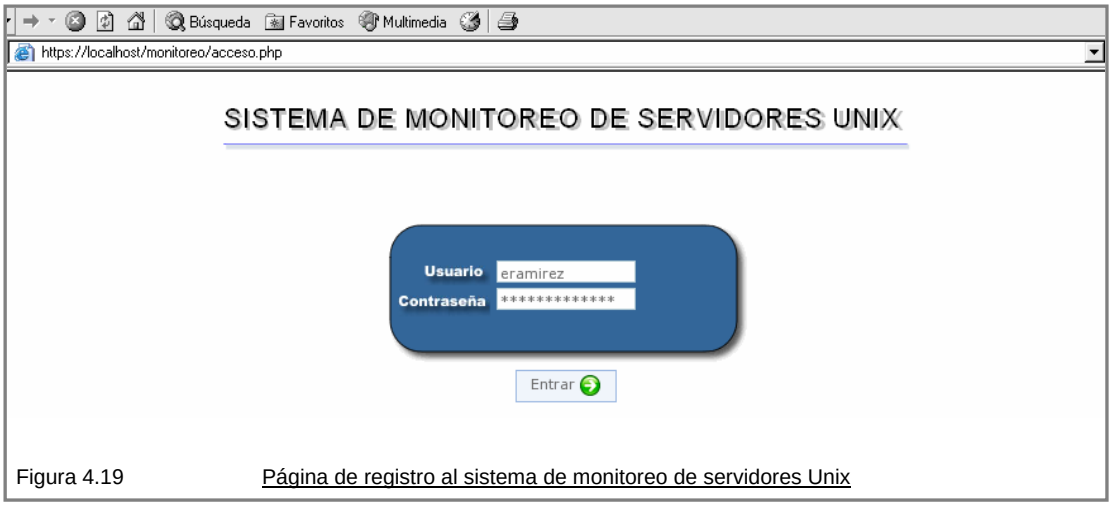


Figura 4.19 Página de registro al sistema de monitoreo de servidores Unix

El sistema responderá con un mensaje de error en caso de que la combinación de usuario y contraseña sea incorrecta (figura 4.20).


Figura 4.20 Mensaje de error de autenticación

En caso de que la combinación de usuario y contraseña sean correctas, el sistema permitirá la entrada al usuario. Una vez dentro del sistema, éste mostrará un menú con tres opciones: Servidores, Catálogos y Administración y sus correspondientes subopciones.

Para el menú “Servidores”, la lista de opciones es: Monitoreo y Reportes, para el menú “Catálogos”, la lista se integra por Correos, Mensajes, Búsquedas, Puertos (estado), Usuarios (estado), Sistemas Operativos y Servidores (estado) y finalmente para el menú “Administración” las opciones de Horarios, Usuarios, Servidores preliminares y Servidores monitoreados figura 4.21.

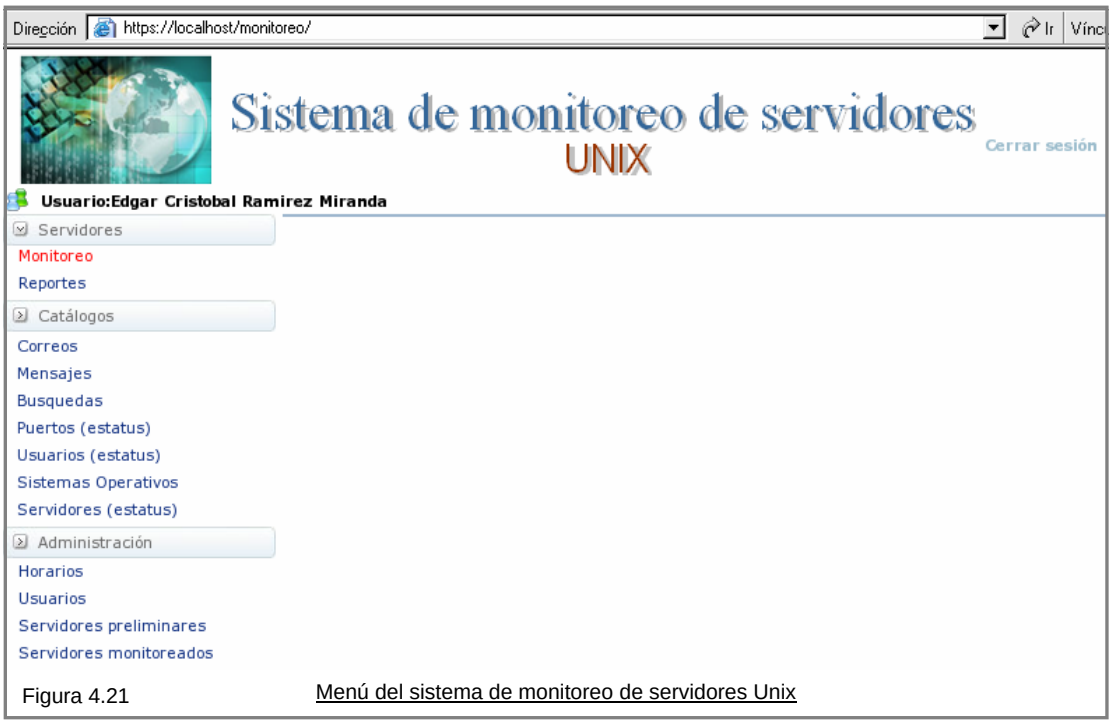


Figura 4.21 Menú del sistema de monitoreo de servidores Unix

Antes de iniciar el monitoreo es recomendable establecer el tiempo de la frecuencia de monitoreo, el cual por defecto es cada 30 segundos, este tiempo se define en el menú de administración en la opción “Horarios” figuras 4.22 y 4.23.


Figura 4.22 Frecuencia de monitoreo

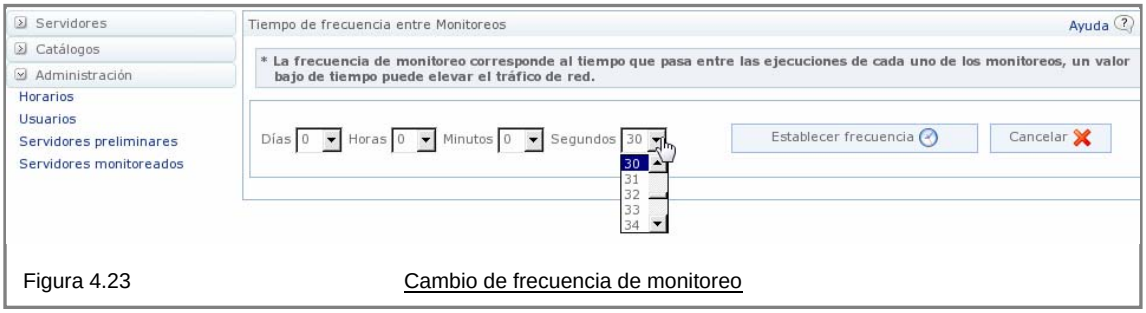


Figura 4.23 Cambio de frecuencia de monitoreo

Una frecuencia baja de monitoreo aumentará el tráfico de red y afectará el desempeño de los servidores monitoreados, una frecuencia de monitoreo demasiado alta no permitirá detectar los errores de manera oportuna. Debido a esto la frecuencia de monitoreo máxima por defecto es de 30 segundos aunque una frecuencia de 1 minuto es recomendable para monitoreos con un promedio de 50 servidores.

### 4.2.4 Ayuda

El sistema esta distribuido de manera tal que la navegación a través de él, no debería representar ningún problema, sin embargo, existe ayuda disponible en cada una de las páginas la cual nos permitirá resolver cualquier duda acerca de la operación de la pantalla en la que nos encontremos.

El ícono de ayuda se encuentra en la parte superior derecha de todas las pantallas adjunto a la palabra “Ayuda”, como se muestra en la figura 4.24.


Figura 4.24 Ícono de Ayuda

Adicionalmente todos los botones y algunos textos cuentan con mensajes explicativos que se despliegan con sólo pasar el puntero sobre ellos y que proporcionan respuesta inmediata a la funcionalidad de estos (figura 4.25).

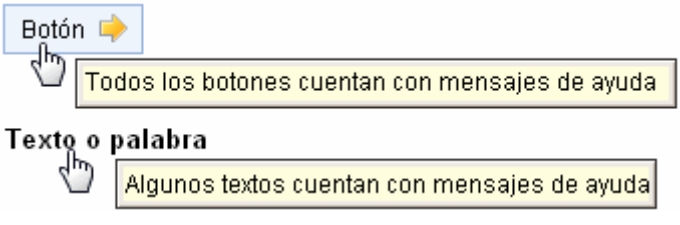


Figura 4.25 Ayuda a través de mensajes

Al presionar el ícono de ayuda se replegara automáticamente una venta con información referente a la pantalla actual, como se muestra en la figura 4.26.

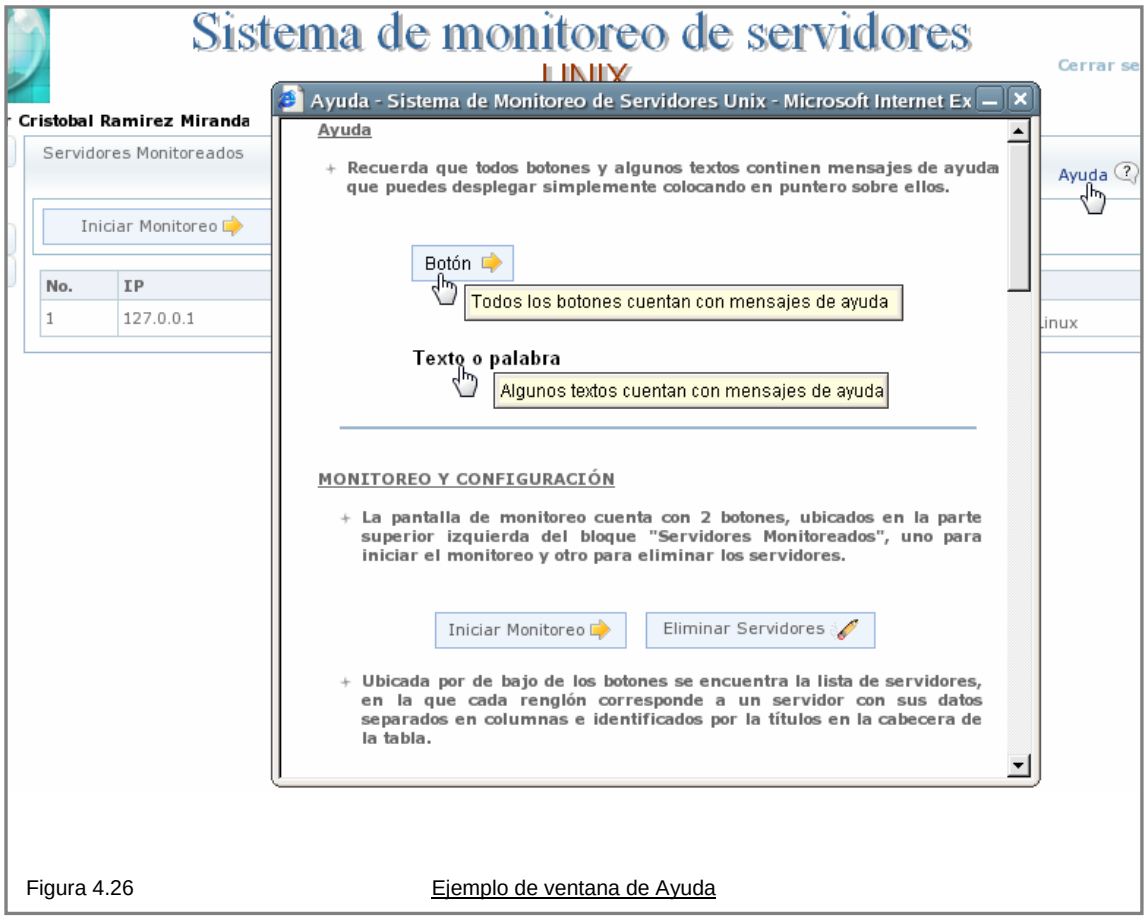


Figura 4.26 Ejemplo de ventana de Ayuda

Adicionalmente es posible encontrar respuestas en la documentación que acompaña a la distribución del Sistema de Monitoreo de Servidores Unix.

### 4.2.5 Perfiles y privilegios (usuarios)

El sistema cuenta con tres perfiles de cuenta de usuario, el administrador, el operador y el usuario normal. El administrador cuenta con todos los privilegios, el operador puede cambiar los parámetros y detener los monitoreo pero no dar de alta usuarios ni servidores, el usuario sólo puede consultar los registros de los servidores pero no modificar ningún parámetro.

El alta de cuentas debe ser realizada por la cuenta de administrador, ya que un principio es el único usuario del sistema. Para crear un usuario se oprime el botón “Nuevo”, esto abrirá una ventana en la cual se solicitarán los datos del usuario y se establecerá su perfil como se muestra en las figuras 4.27 y 4.28.

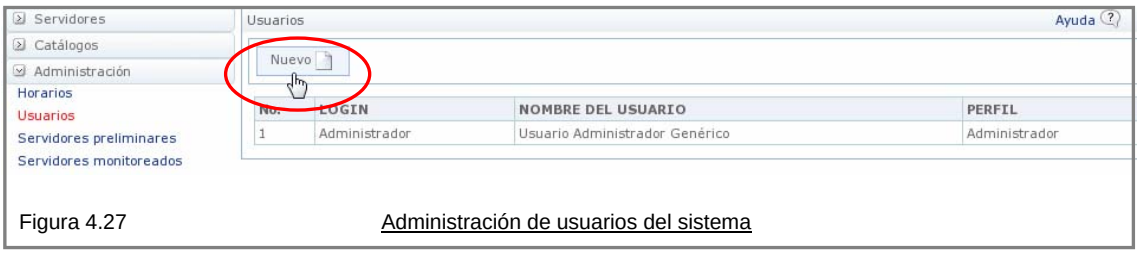


Figura 4.27 Administración de usuarios del sistema


Figura 4.28 Administración de usuarios (alta)

De manera similar los datos de los usuarios ya registrados pueden ser consultados o modificados como se muestra en la figura 4.29.

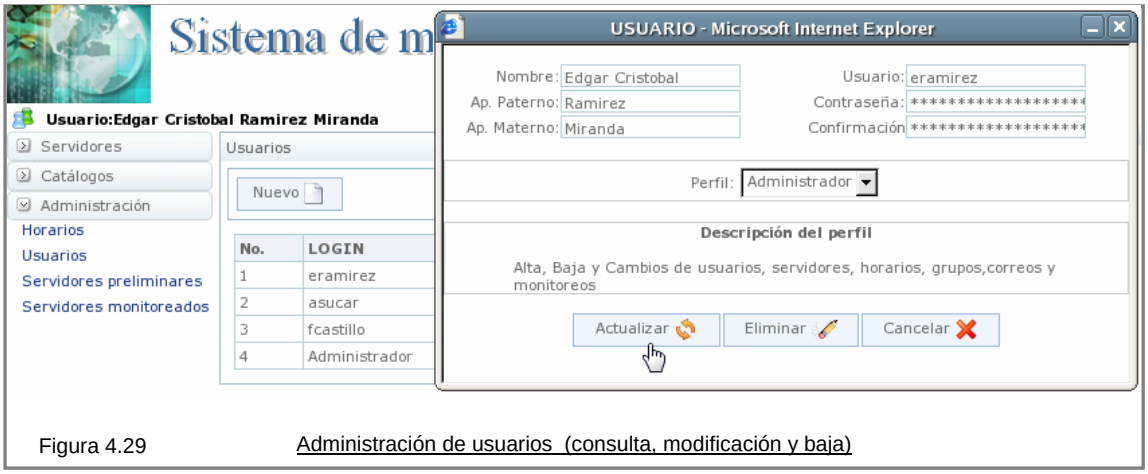


Figura 4.29 Administración de usuarios (consulta, modificación y baja)

### 4.2.6 Visualización de datos de monitoreo

Los servidores a monitorear son dados de alta a través de un proceso de detección en el cual se verifica que el servidor este arriba o en línea y que pueda ser identificado de manera correcta, para este proceso se utiliza la opción “Servidores preliminares” del menú de “Administración”. Apretando el botón “Nuevo” aparecerá una ventana en la cual se ingresarán los datos del servidor a detectar (El sistema puede también agregar servidores Windows pero sólo para la detección, es decir, sólo reportará que éstos están en línea. Figuras 4.30 y 4.31).


Figura 4.30 Registro preliminar de servidores

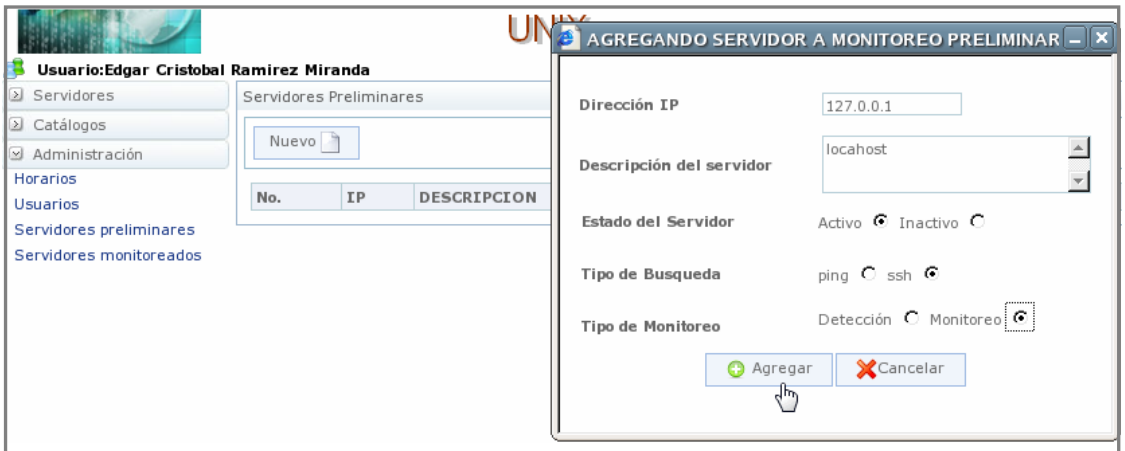


Figura 4.31 Registro preliminar de servidores

Una vez registrado el servidor y definido el método de monitoreo, se puede oprimir el botón “Detectar”, el cual tratará de detectar al servidor y determinar el sistema operativo de que se trate (figura 4.32).

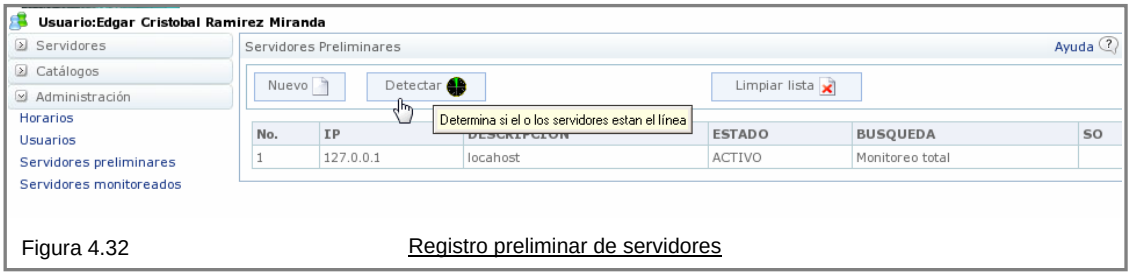


Figura 4.32 Registro preliminar de servidores

Si la detección del servidor fue exitosa, se presentará la leyenda “ACTIVO Detección exitosa” en la columna de “ESTADO” y el nombre del sistema operativo en la columna SO (Sistema Operativo) a su vez aparecerá un el botón “Agregar al monitoreo”, el cual permitirá que el servidor sea ingresado al ciclo de monitoreo (figura 4.33). Este proceso inicializa los datos del servidor y ahora su seguimiento se llevará en la opción de “Servidores monitoreados”.


Figura 4.33 Agregar servidor preliminar al monitoreo

El servidor detectado aparecerá en la lista de servidores monitoreados, sin embargo el proceso de monitoreo puede no estar activado, este proceso se inicia al apretar el botón “Iniciar monitoreo” (figura 4.34).


Figura 4.34 Lista de servidores monitoreados

Al apretar el botón “Iniciar Monitoreo”, este cambiará por el botón “Detener Monitoreo”, con el cual se puede detener de manera general el proceso de monitoreo de todos los servidores en cualquier momento (figura 4.35).


Figura 4.35 Monitoreando servidores

Durante el proceso de monitoreo, la lista de servidores se actualizará para mostrar estado actual de cada uno de los servidores, en caso de detectarse una alarma, el color del renglón del servidor correspondiente cambiará a rojo desde el momento en que la alarma se detecte hasta que ésta se corrija (figura 4.32).


Figura 4.36 Alarma detectada durante el monitoreo

Los datos obtenidos en el monitoreo podrán ser consultados dando un clic sobre el renglón del servidor correspondiente (figura 4.37).

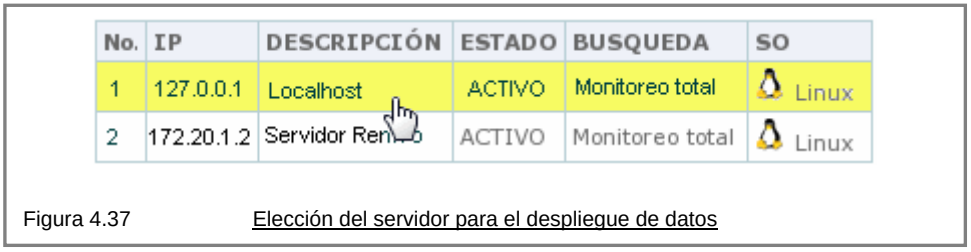


Figura 4.37 Elección del servidor para el despliegue de datos

Los datos se mostrarán en bloques de información de la misma categoría (figura 4.38).

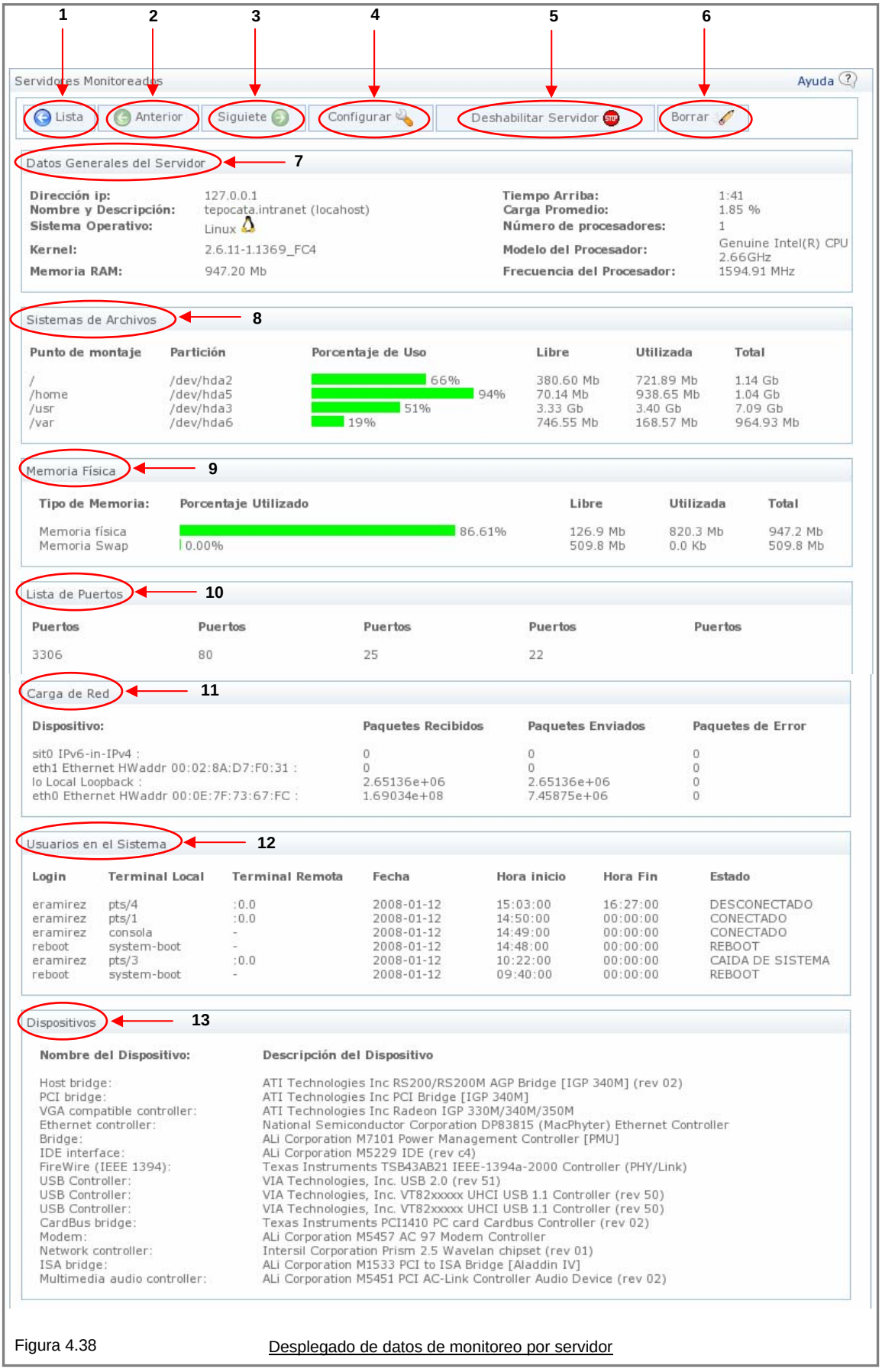


Figura 4.38 Desplegado de datos de monitoreo por servidor

En la figura 4.38 podemos identificar los siguientes componentes:

Botones

1. Lista - Este botón nos regresa a la lista general de servidores.
2. Anterior - Nos muestra los datos de monitoreo del servidor anterior.
3. Siguiente - Nos muestra los datos de monitoreo del siguiente servidor.
4. Configurar - Abre la ventana de configuración del servidor correspondiente.
5. Deshabilitar servidor/ Activar servidor - Este botón activa y desactiva el monitoreo del servidor de manera individual.
6. Borrar - Este botón borra al servidor y todos sus datos de manera individual.

Módulos

7. Datos generales del servidor

Este módulo presenta los datos generales del servidor monitoreado, tales como su dirección ip, nombre y descripción, sistema operativo, kernel, memoria ram, tiempo que lleva encendido el servidor, carga promedio, número de procesadores, modelo y frecuencia del procesador o procesadores.

8. Sistemas de archivos

Este módulo presenta el punto de montaje, partición, porcentaje de utilización, espacio libre, espacio utilizado y tamaño total de los sistemas de archivos del servidor.

9. Memoria física

Este módulo muestra el tamaño total, el espacio libre y el utilizado y el porcentaje de utilización de las memorias ram y swap.

10. Lista de puertos

Este módulo presenta una lista de todos los puertos activos en servidor, estos puertos pueden ser configurados para ser monitoreados (por defecto todos los puertos están etiquetados como tolerados, lo que significa que no se reporta nada en caso de que estos dejen de estar habilitados), cabe destacar que este módulo detecta activaciones de puertos no autorizados.

11. Carga de red

Este módulo presenta la carga (paquetes enviados, recibidos y paquetes de error) en las diferentes interfases de red del servidor.

12. Usuarios en el sistema

Este módulo muestra un lista de los usuarios del sistema con un formato similar al del comando last. Este listado separa los registros de los usuarios en, login o nombre de la cuenta de usuario, terminal local (terminal a la que se conectó el usuario en el servidor), terminal remota (desde donde se conecto el usuario), fecha de inicio de la conexión (hora, minuto y segundo), fecha de fin de la conexión (hora, minuto y

segundo) y estado actual de la sesión de la cuenta de usuario (Conectado, desconectado, etc.).

13. Dispositivos

Finalmente este módulo presenta una lista de dispositivos del servidor (nombre y descripción del dispositivo).

### 4.2.7 Alarmas

Como ya se mencionó el sistema indicará por medio de un cambio de color cualquier situación detectada (figura 4.36). Todo evento registrado se reportará también dentro del desplegado de datos de monitoreo del servidor correspondiente (figura 4.39), así como en la ventana de configuración de ese parámetro (figura 4.40 ó 4.41). La pantalla de reportes también registrará y mostrará el evento detectado (figura 4.42), finalmente al servidor también se le pueden asignar contactos (correos electrónicos), a los que se les enviará una notificación del evento encontrado vía correo electrónico (figura 4.45).

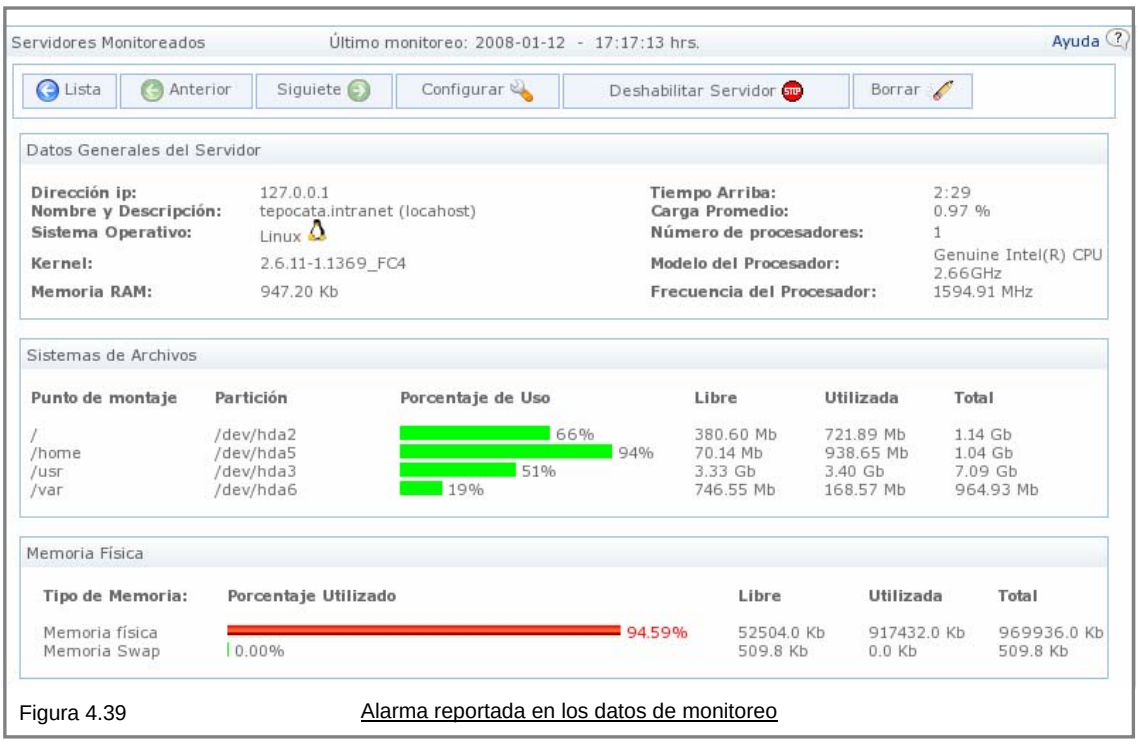


Figura 4.39 Alarma reportada en los datos de monitoreo

Cada uno de los servidores monitoreados puede ser configurado en sus parámetros por medio del botón “Configuración”, el cual al ser presionado despliega una ventana con pestañas en las cuales se pueden configurar los valores de ciertos parámetros de monitoreo (figuras 4.40 y 4.41).

Por ejemplo en la pestaña de generales se puede establecer el valor máximo del porcentaje de utilización del memoria ram y/o de la memoria swap, también se puede establecer el valor limite de la carga del CPU y/o actualizar la descripción del servidor.

En la pestaña de puertos se puede definir si un puerto será tolerado, monitoreado o inactivo.

En la pestaña de sistemas de archivos se puede definir los valores limites de utilización para cada uno de los sistemas de archivos.

En la pestaña de contactos se pueden asignar los correos a los cuales se enviaran a las alarmas en caso de presentarse.

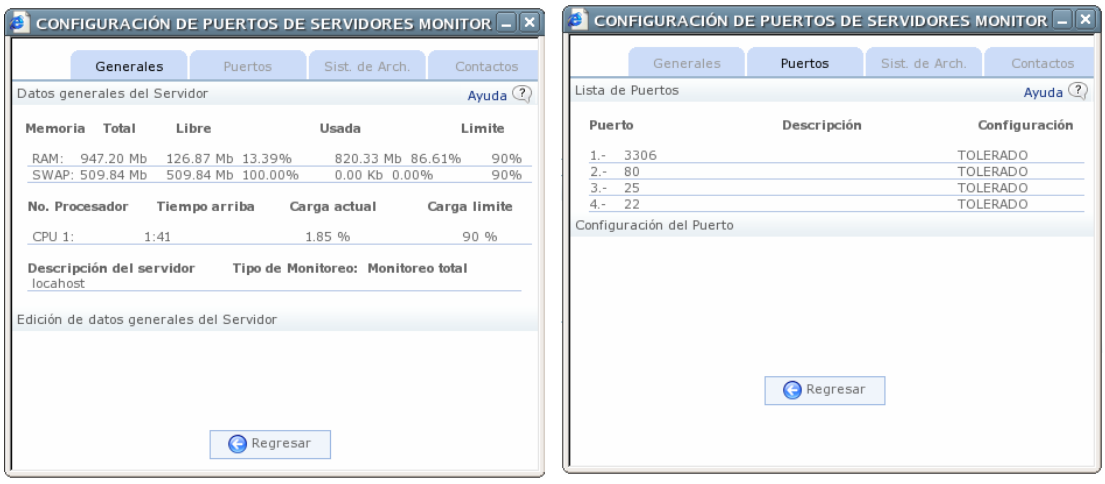


Figura 4.40 Configuración de parámetros

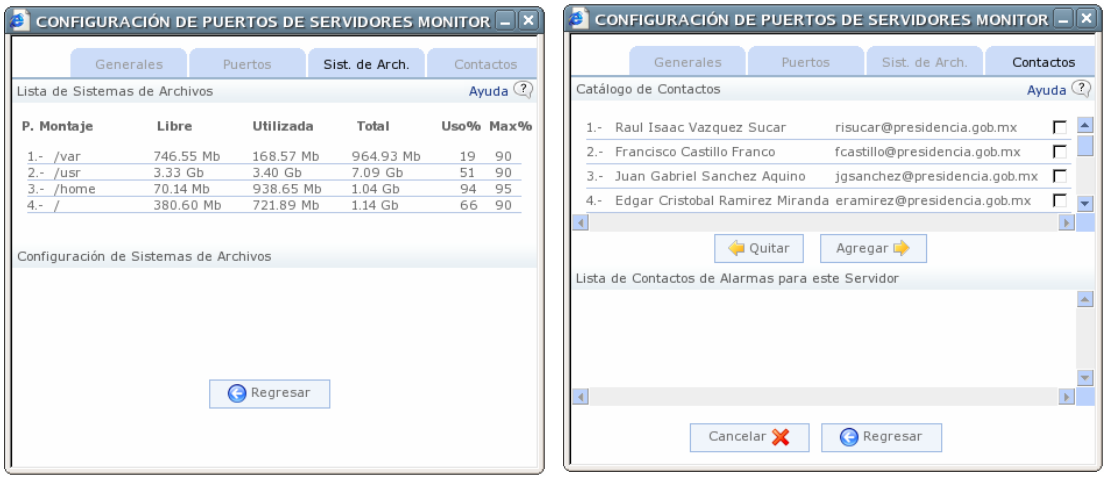


Figura 4.41 Configuración de parámetros

Las alarmas también se pueden visualizar a través de la opción de “Reportes” como se muestra en la figura 4.42. Las alarmas se muestran en un listado de renglones y el detalle o configuración actual de la alarma relacionada puede ser desplegado al dar un clic en el renglón correspondiente.


Figura 4.42 Reporte de alarmas

La pantalla de reportes cuenta con filtros que permiten desplegar la información de manera selectiva. Por defecto la pantalla de reportes muestra los eventos de la fecha actual, es decir, el filtro “Hoy”. El filtro “Últimos” desplegará la última ocurrencia de cada uno de los diferentes eventos para cada uno de los servidores, es decir, eventos del mismo tipo que se repitan solo serán representados por su última ocurrencia. También es posible acotar la búsqueda de eventos a un rango de fechas y horarios a través del filtro “Especificar fechas y horas” como muestra en la figura 4.43.

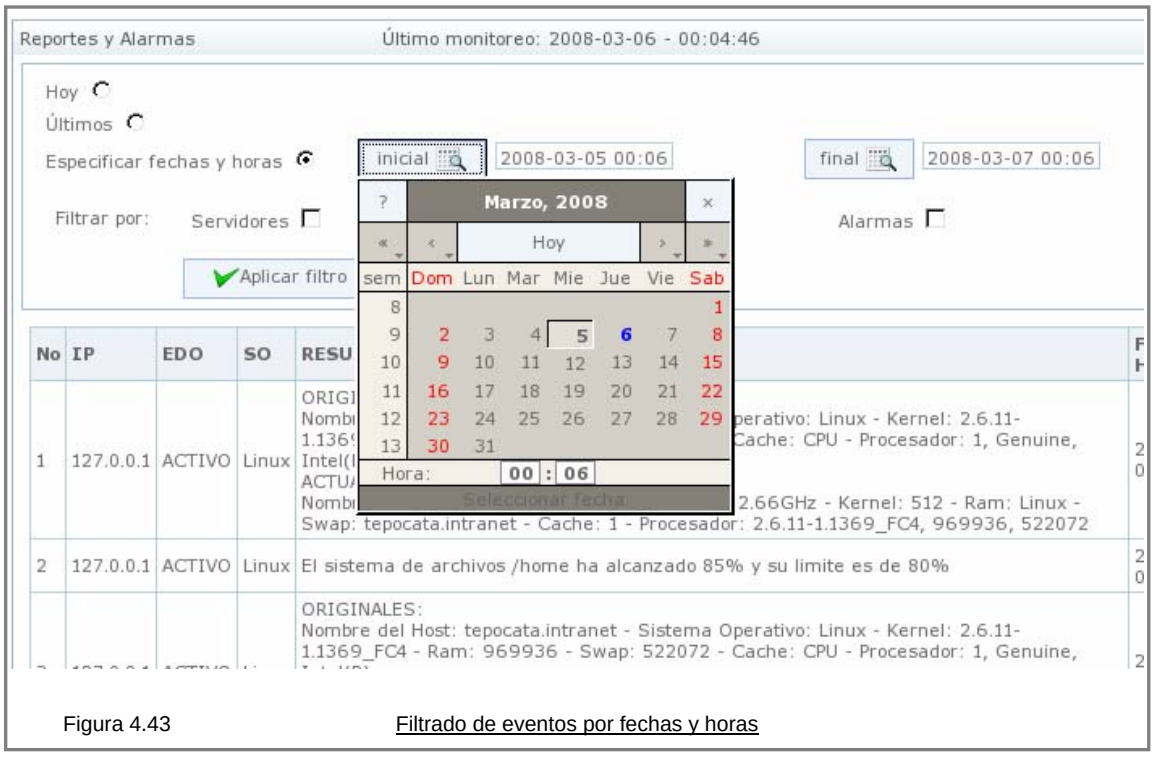


Figura 4.43 Filtrado de eventos por fechas y horas

También es posible extraer información en el rango de tiempo definido de servidores o alarmas en específico, utilizando los filtros “Servidores” y “Alarmas” como se muestra en la figura 4.44.

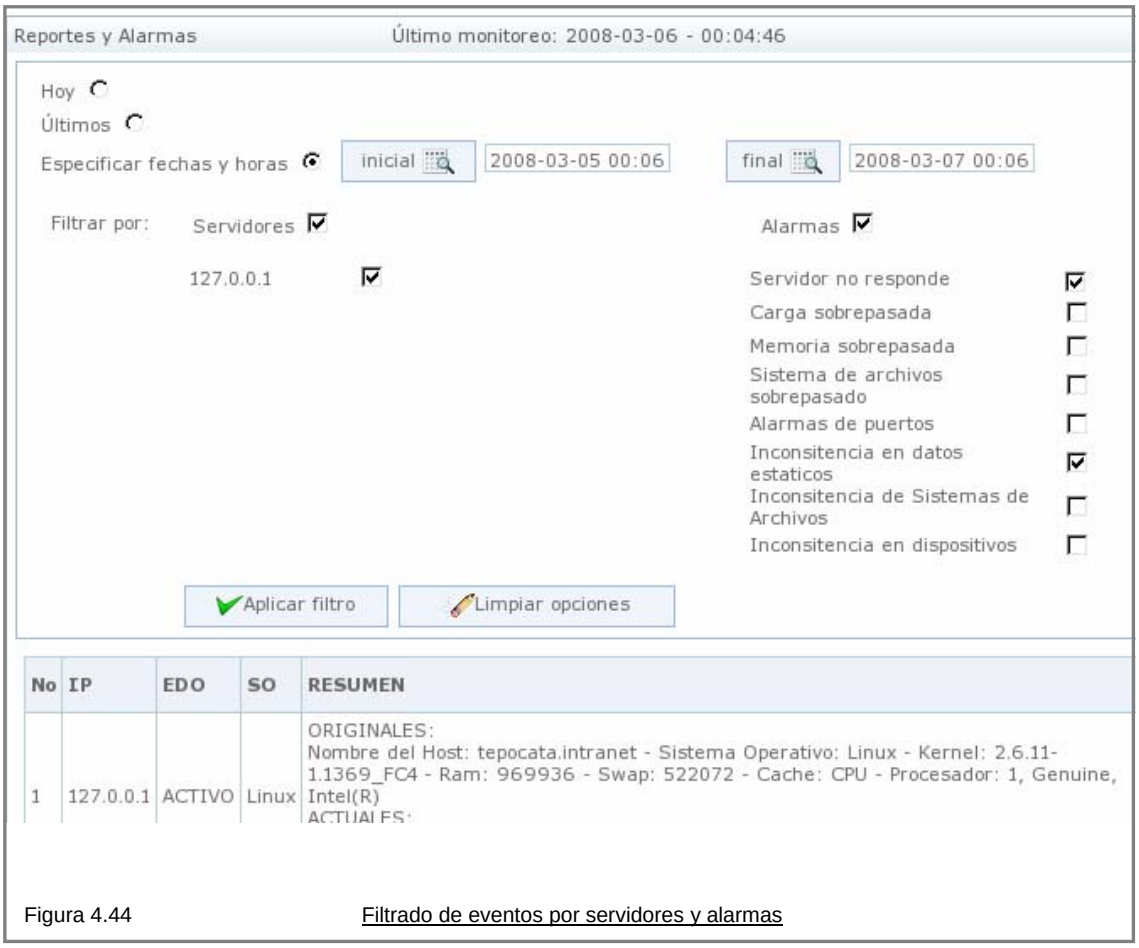


Figura 4.44 Filtrado de eventos por servidores y alarmas

Por último el sistema de monitoreo es capaz de enviar el reportes de las alarmas vía correo electrónico, como se muestra en la figura 4.45.

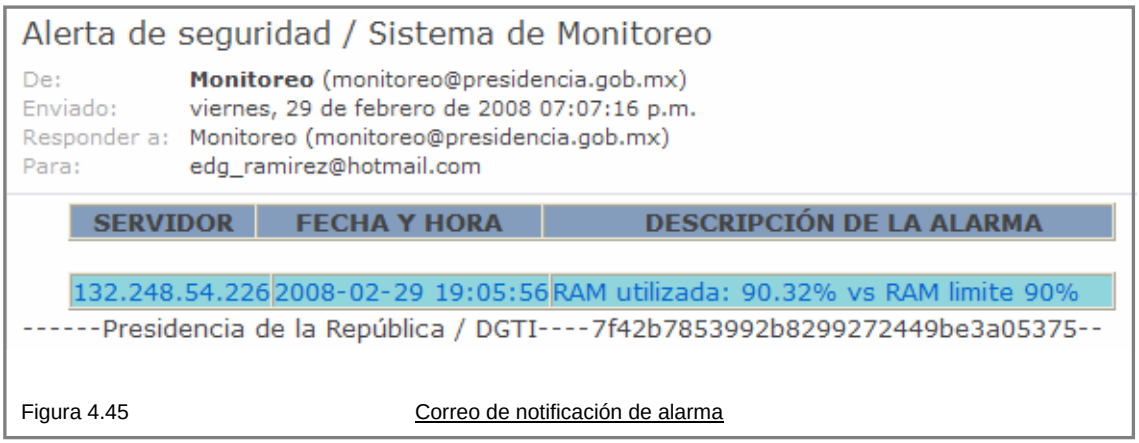


Figura 4.45 Correo de notificación de alarma

### 4.2.8 Salida del sistema

Para salir del sistema se utiliza el letrero de “Cerrar sesión”, el cual siempre esta presente en la parte superior izquierda de la pantalla, tal y como se muestra en la figura 4.46.


Figura 4.46 Salida del sistema

Este sistema fue implementado con éxito en una organización gubernamental y en la Facultad de Ingeniería.

### 4.2.9 Costos

El costo total del sistema es un poco complejo de estimar o determinar, principalmente debido a dos situaciones:

- El tiempo real dedicado al análisis, desarrollo e implementación del sistema no fue uniforme ni constante derivado de cuestiones laborales que impidieron su continuidad y seguimiento adecuado.
- El análisis de costos se determina en parte a los recursos utilizados para terminar y entregar el producto final (instalaciones, personal, etc.) en este caso el sistema requirió de dos perfiles de personal un administrador y un programador o un programados con conocimientos de sistemas operativos, lo que en la práctica es difícil de encontrar.

El sueldo promedio de un programador de PHP con conocimientos de bases de datos fluctúa entre los $8,000 y los $12,000 pesos, mientras que el sueldo promedio de un administrador o programador de scripts fluctúa entre los $14,000 y $26,000 pesos.

El sueldo promedio de estos perfiles es de $10,000 y $20,000 pesos respectivamente y el promedio de ellos es de $15,000, siendo este último el tomado como referencia para la estimación del costo total.

El tiempo efectivo dedicado para el análisis e implementación del proyecto se estima de 5 a 6 meses, por lo que el costo total se encuentra entre $75,000 y $90,000 pesos MN al 28 de febrero de 2008.

### 4.2.10 ¿Hacia dónde vamos?

El sistema de monitoreo esta diseñado tal manera que nuevos módulos de monitoreo pueden ser agregados con facilidad, tales como módulos para medir la respuesta de servicios como POP, http, bases de datos, procesos, etcétera.

Hoy en día es cada vez más tangible la necesidad de sustituir los esquemas de comunicación inseguros por sus contrapartes seguras, lo mismo sucede a nivel de switches y ruteadores, los cuales incluyen ahora soporte para ssh en lugar de Telnet, con lo cual también se abre la posibilidad de monitorearlos con la aplicación aquí desarrollada.

# Conclusiones y comentarios finales

## Conclusiones y comentarios finales:

Al finalizar el presente trabajo de tesis, se lograron los objetivos inicialmente establecidos:

- Se diseñó, desarrolló e implementó un sistema de monitoreo configurable que permite obtener información valiosa de los servidores Unix y Linux y de los procesos que sobre ellos se ejecutan, para garantizar la operación óptima y en su caso actuar con oportunidad con acciones correctivas.

- Se implemento el sistema de monitoreo basado en conceptos de seguridad que permiten tener un nivel alto de confiabilidad en cuanto a la seguridad de la información que se obtiene de los servidores.

- Se creo un sistema de monitoreo con un entorno amigable e intuitivo para su administración y operación.

- A través del Sistema de Monitoreo de Servidores Unix, es posible identificar patrones de comportamiento anómalos, potencialmente dañinos o simplemente representativos del comportamiento y que pudieran ser utilizados para la toma de decisiones.

Al realizar un análisis de diferentes herramientas libres de monitoreo se encontraron deficiencias de seguridad en todas ellas, motivo por el cual se tomó la decisión de desarrollar una herramienta que cumpliera las necesidades particulares de seguridad y las funcionalidades generales a una herramienta de monitoreo de servidores. El modelo bajo el cual se desarrolló la aplicación fue el modelo incremental, el cual combina elementos de modelo lineal secuencial con la filosofía iterativa del modelo de construcción de prototipos.

La herramienta desarrollada esta estructurada de manera modular alrededor de un motor principal de monitoreo, al cual se le pueden agregar módulos para incrementar la funcionalidad tanto como se requiera, opera en un esquema centralizado sin la utilización de clientes de monitoreo por lo que los puntos de contacto y posibles vulnerabilidades relativas a los clientes se abaten al sólo tener que concentrar la seguridad en un punto. Parte de la seguridad se logra en la forma de extraer los datos, la cual incluye confidencialidad e integridad por medio de alto cifrado y autenticación tanto del servidor monitoreado como de la cuenta de usuario utilizada para ejecutar el monitoreo, todo esto a través de un mecanismo de par de llaves. Se utiliza un canal o puerto de administración estándar por lo cual no se requiere de permisos adicionales para la implementación de la aplicación. La extracción de datos de los servidores se realiza desde adentro de los propios servidores con lo cual se evitan las pruebas externas que podrían en algún momento ser consideradas intrusivas.

El proyecto de “Monitoreo de Servidores Unix” presentado en esta tesis atiende a las necesidades particulares de monitoreo cuyos requerimientos de seguridad son más altos que los implementados en las herramientas de monitoreo de servidores comunes, más sin embargo, el sistema de monitoreo desarrollado puede ser fácilmente utilizado en cualquier entorno de servidores Unix o tipo Unix proporcionando el valor agregado del alto nivel de seguridad proporcionado por la metodología empleada en el monitoreo sin menoscabo de la velocidad en la extracción de datos.

Este sistema fue probado e implementado con éxito en una entidad del gobierno y en la Facultad de Ingeniería. Actualmente se encuentra operando en el departamento de redes y operación de servidores de UNICA.

# Apéndices

**Licencia BSD**

Es la licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Pertenece al grupo de licencias de software libre. Esta licencia tiene menos restricciones en comparación con otras como la GPL estando muy cercana al dominio público. La licencia BSD al contrario que la GPL permite el uso del código fuente en software no libre.

La licencia tiene el siguiente formato.

```
/*-
 * Copyright (c) <año> <autores>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of copyright holders nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
 * TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL COPYRIGHT HOLDERS OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
```

El autor, bajo esta licencia, mantiene la protección de copyright únicamente para la renuncia de garantía y para requerir la adecuada atribución de la autoría en trabajos derivados, pero permite la libre redistribución y modificación.

Puede argumentarse que esta licencia asegura “verdadero” software libre, en el sentido que el usuario tiene libertad ilimitada con respecto al software, y que puede decidir incluso redistribuirlo como no libre. Otras opiniones están orientadas a destacar que este tipo de licencia no contribuye al desarrollo de más software libre.

**PhpMysAdmin**

A continuación se describen las características de phpMyAdmin a través de las siguientes imágenes:

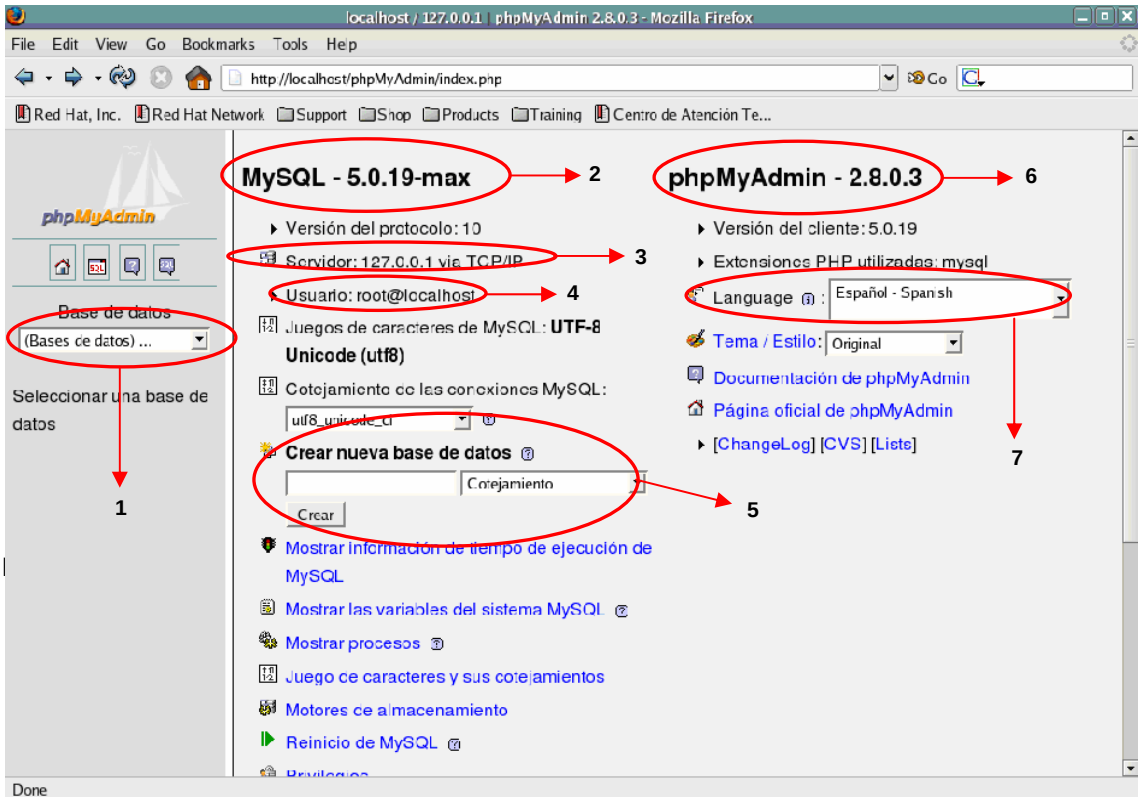


Figura A.1 Página de inicio de phpMyAdmin

1.- Lista de bases de datos.
2.- Versión del servidor MySQL.
3.- Servidor al que esta conectado.
4.- Usuario con el que se conecta.
5.- Apartado por medio del cual se crean las bases de datos.
6.- Versión de phpMyAdmin.
7.- Idioma de la Interfase.

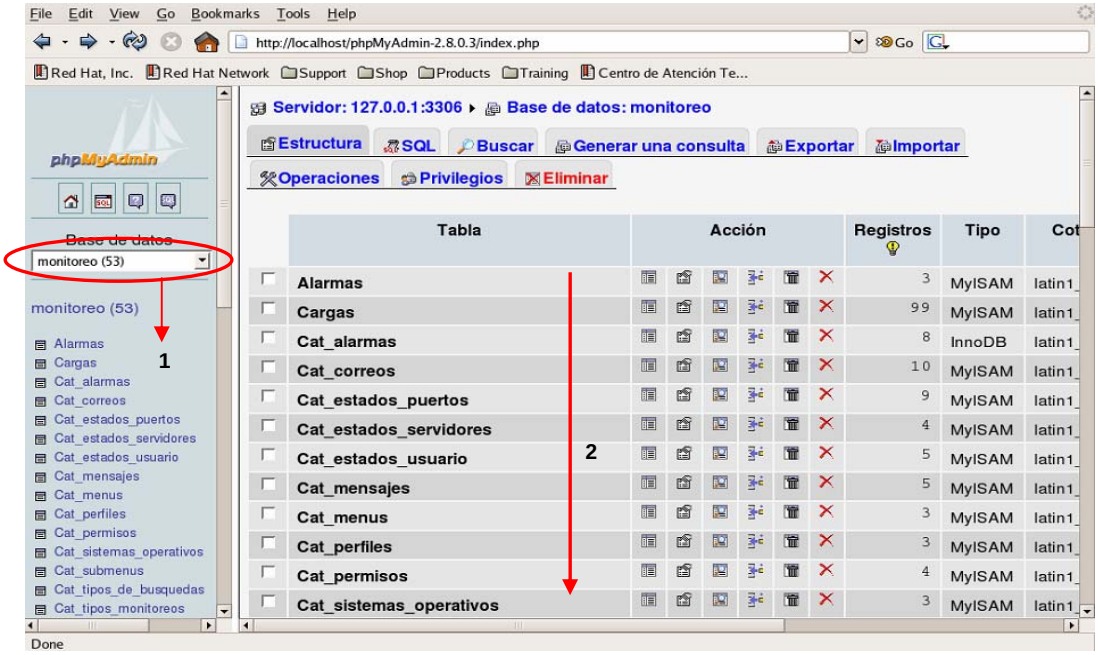


Figura A.2 Lista de tablas de la base de datos

La figura A.2 muestra las posibles operaciones sobre una base de datos:

1. Al seleccionar por medio de la lista de bases de datos una base de datos en particular, se mostraran las tablas que componen a esta base de datos, tanto en el recuadro de la izquierda, como en el recuadro de la derecha en donde se da más detalle de cada una de ellas.
2. La lista detallada de las tablas, permite realizar operaciones sobre ellas (eliminarlas, alterarlas).

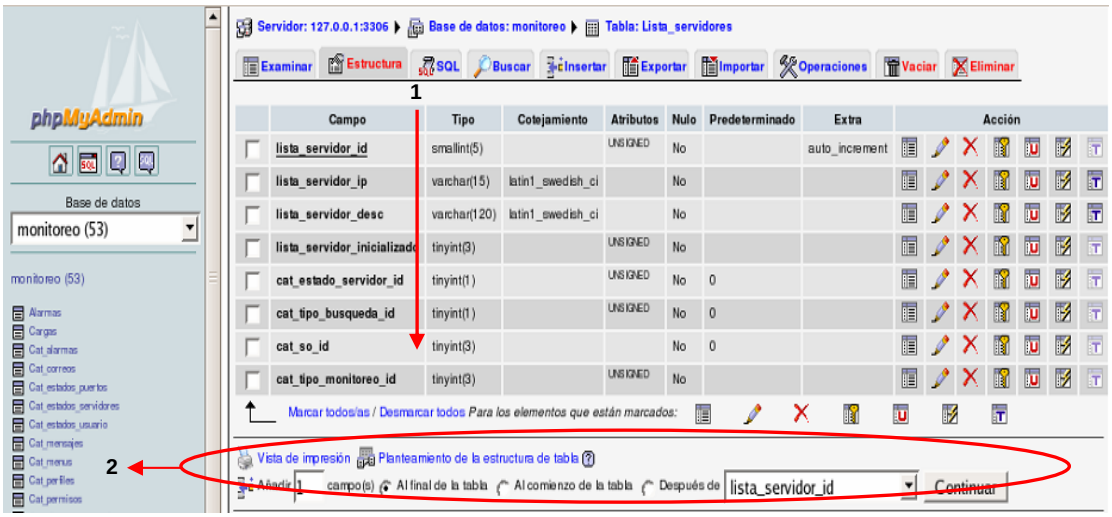


Figura A.3 Vista de los campos de una tabla en particular

La figura A.3 muestra las posibles operaciones sobre los campos de una tabla:

1. Al seleccionar una tabla en particular se muestran sus campos, con información del tipo de datos que cada uno de ellos guarda. Estos campos a su vez pueden ser alterados o eliminados.
2. También se pueden registrar nuevos campos para la tabla.

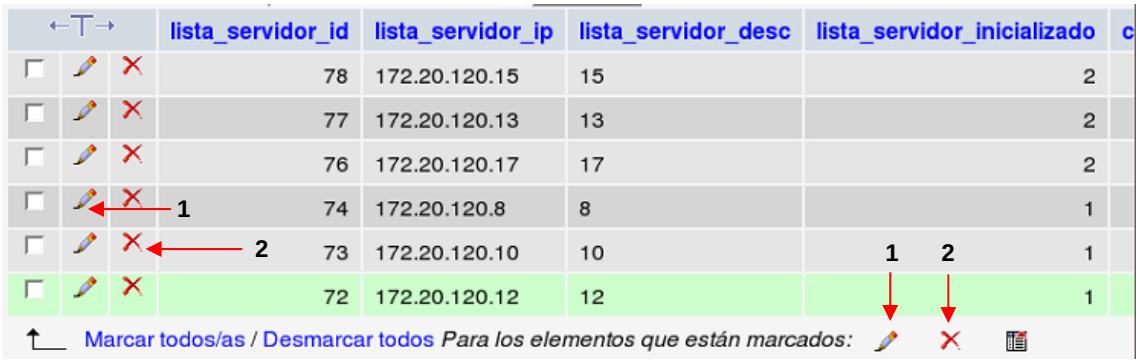


Figura A.4 Registros en una tabla

La figura A.4 muestra las posibles operaciones sobre un registro:

1.- Los registros pueden ser editados, ya sea de manera individual o grupal.
2.- Los registros pueden ser eliminados, ya sea de manera individual o grupal.

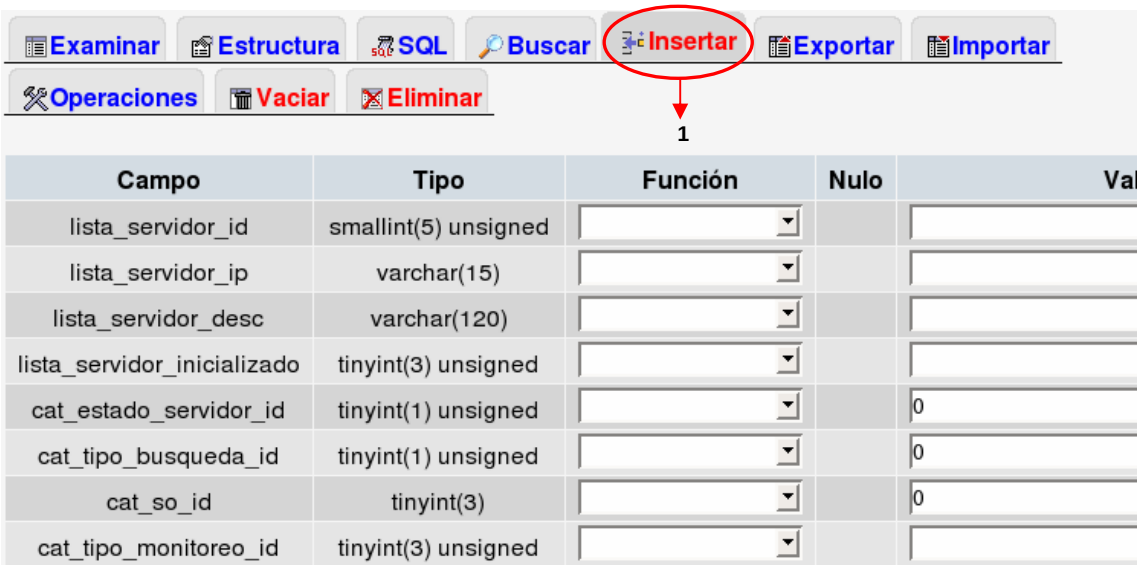


Figura A.5 Inserción de registros de una tabla

1. La inserción de registros se lleva acabo por medio de la opción Insertar, en al cual se muestran recuadros de captura para cada uno de los campos de la tabla, según sea el caso (figura A.5).

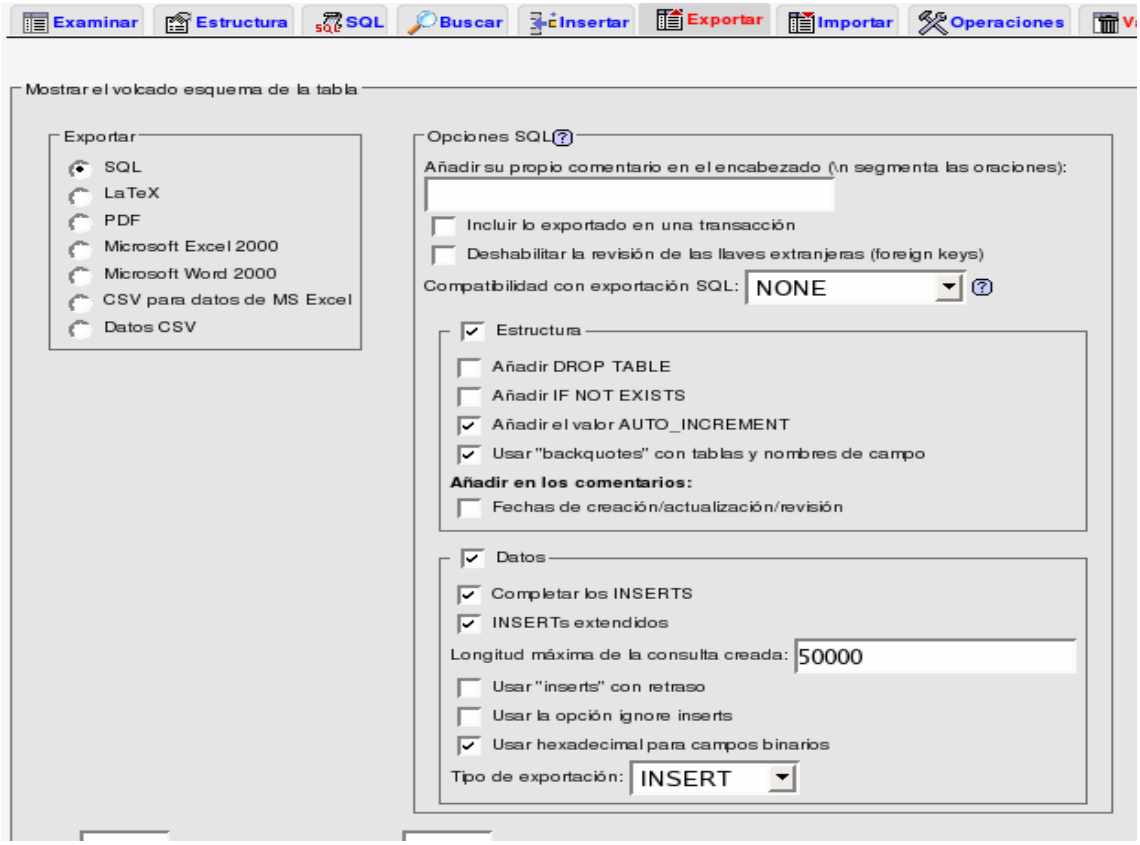


Figura A.6 Exportación de bases de datos

La exportación de la base de datos se lleva acabo a través de la opción exportar, en ella se puede seleccionar de entre varios formatos a exportar (figura A.6).

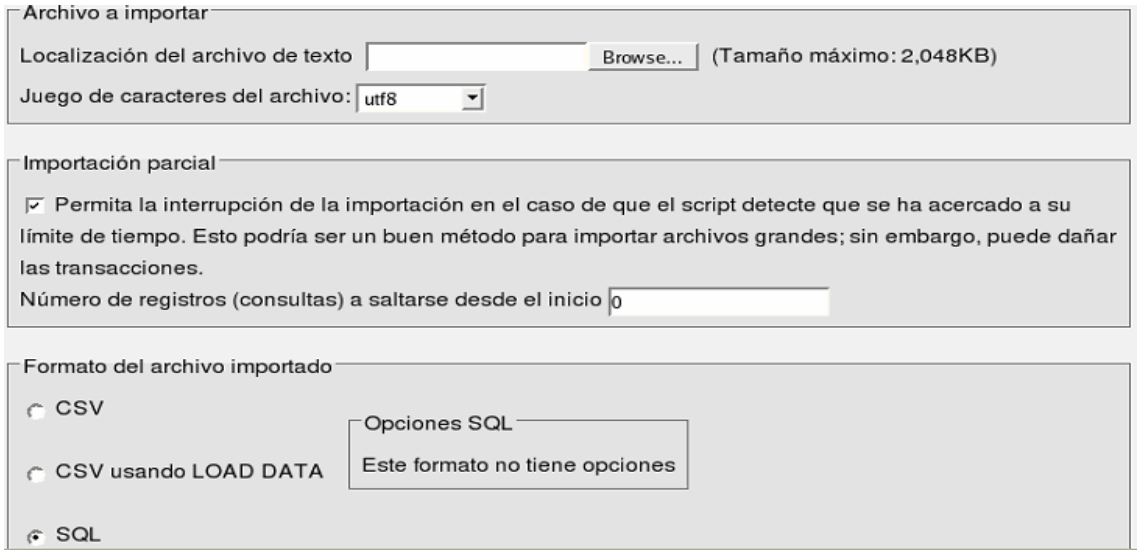


Figura A.7 Importación de bases de datos

La importación de datos se puede llevar a cabo a través de la opción de importar y puede proceder de diferentes tipos de fuentes (figura A.7).

**Diagrama entidad – relación de la base de datos del "Sistema de Monitoreo de Servidores Unix"**

La imagen A.8 muestra el mapa del diagrama entidad relación de la base de datos del sistema de monitoreo de servidores Unix. Como se puede apreciar, el diagrama entidad-relación es muy grande como para poder apreciarlo en una sola vista, por lo que se secciono en partes más pequeñas las cuales se presentan en las figuras A.9, A.10, A.11, A.12, A.13, A.14, A.15 y A.16.

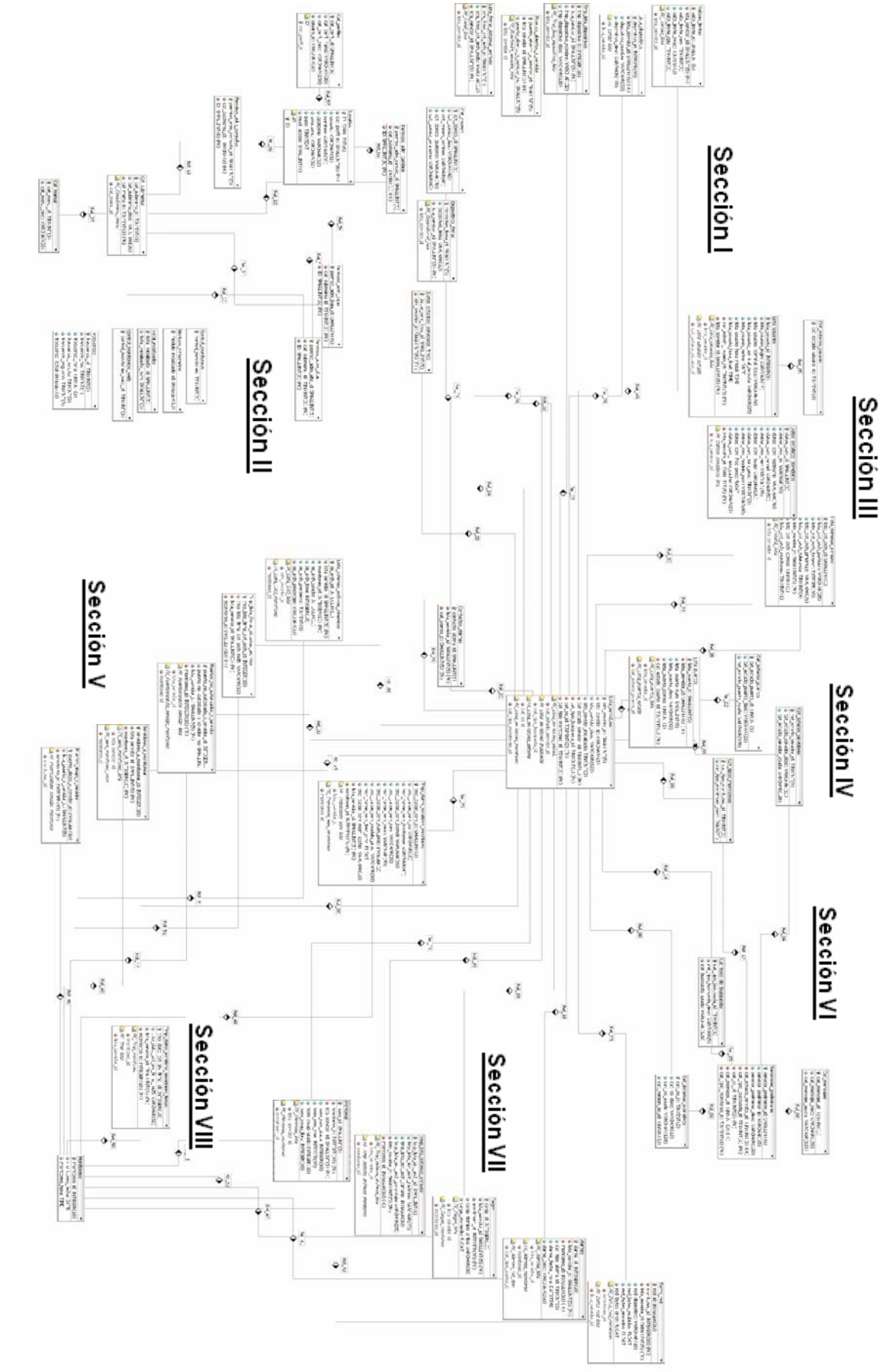


Figura A.8 Mapa del diagrama entidad –relación de la base de datos

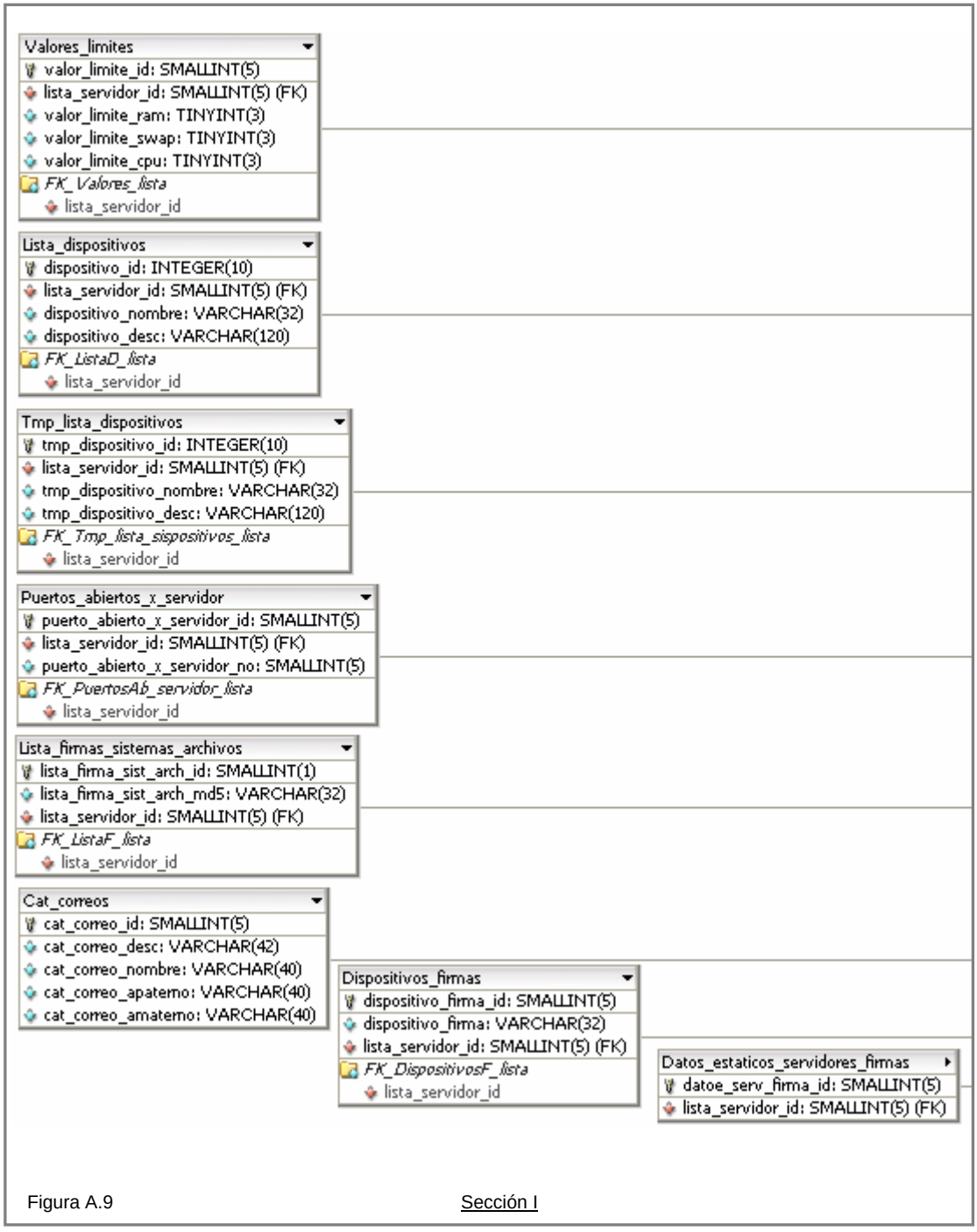


Figura A.9 Sección I

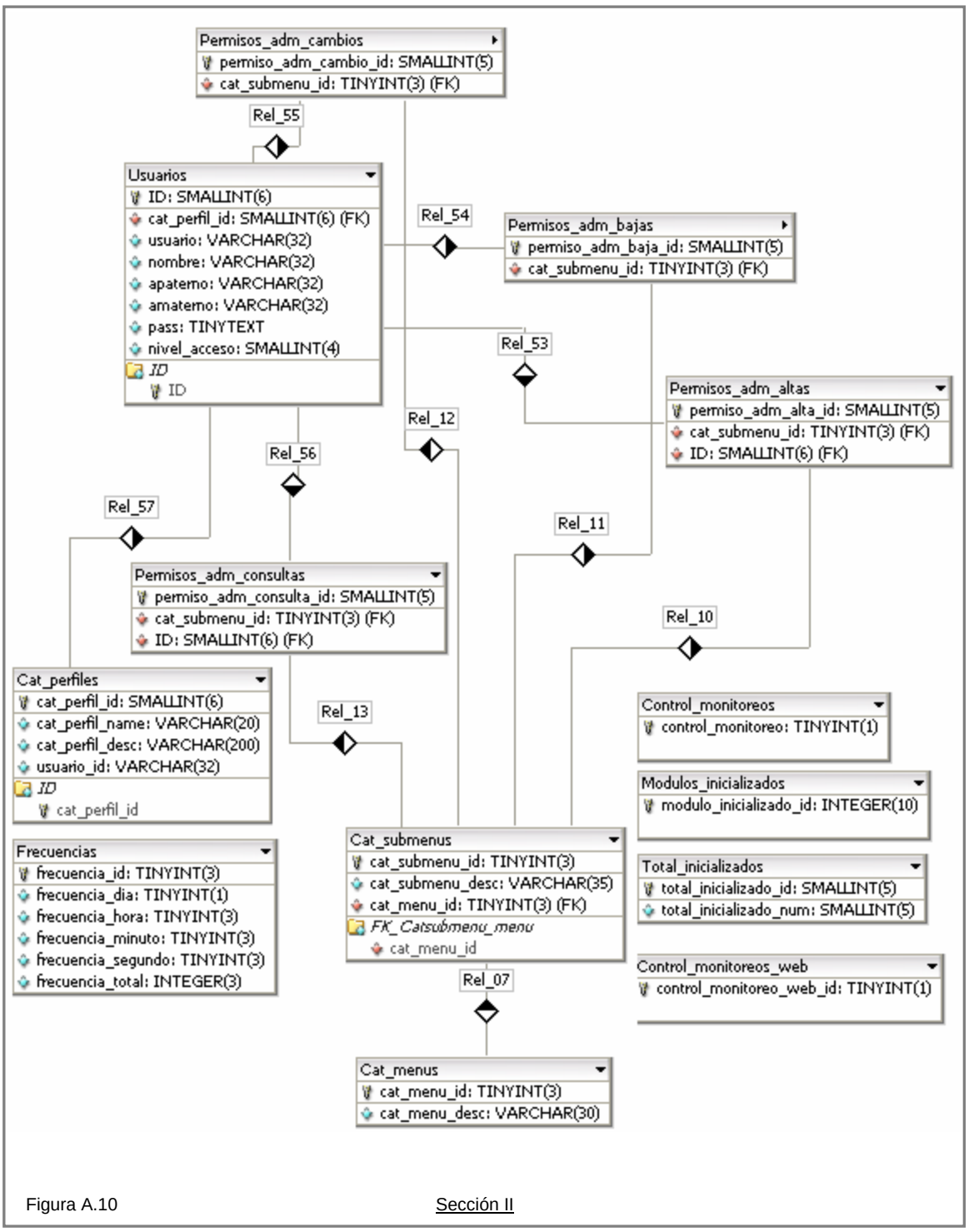


Figura A.10 Sección II

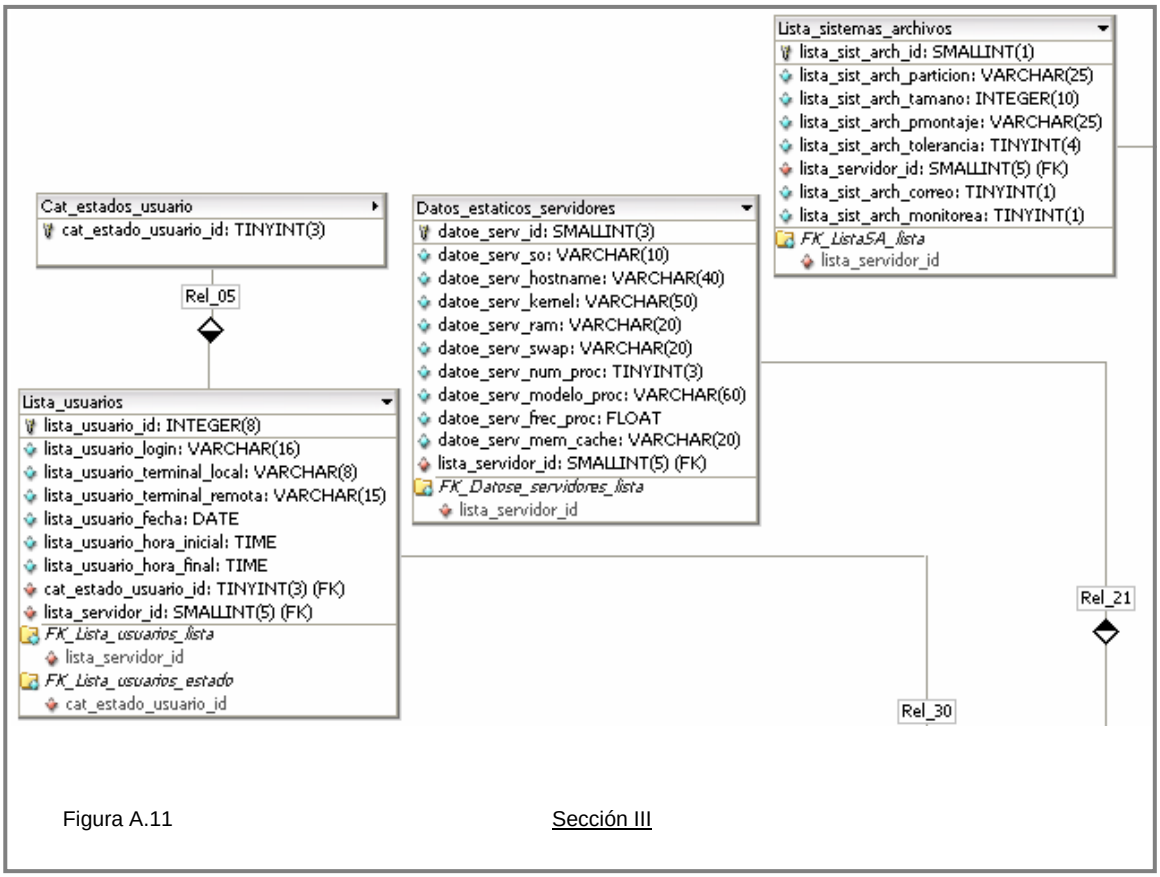


Figura A.11 Sección III

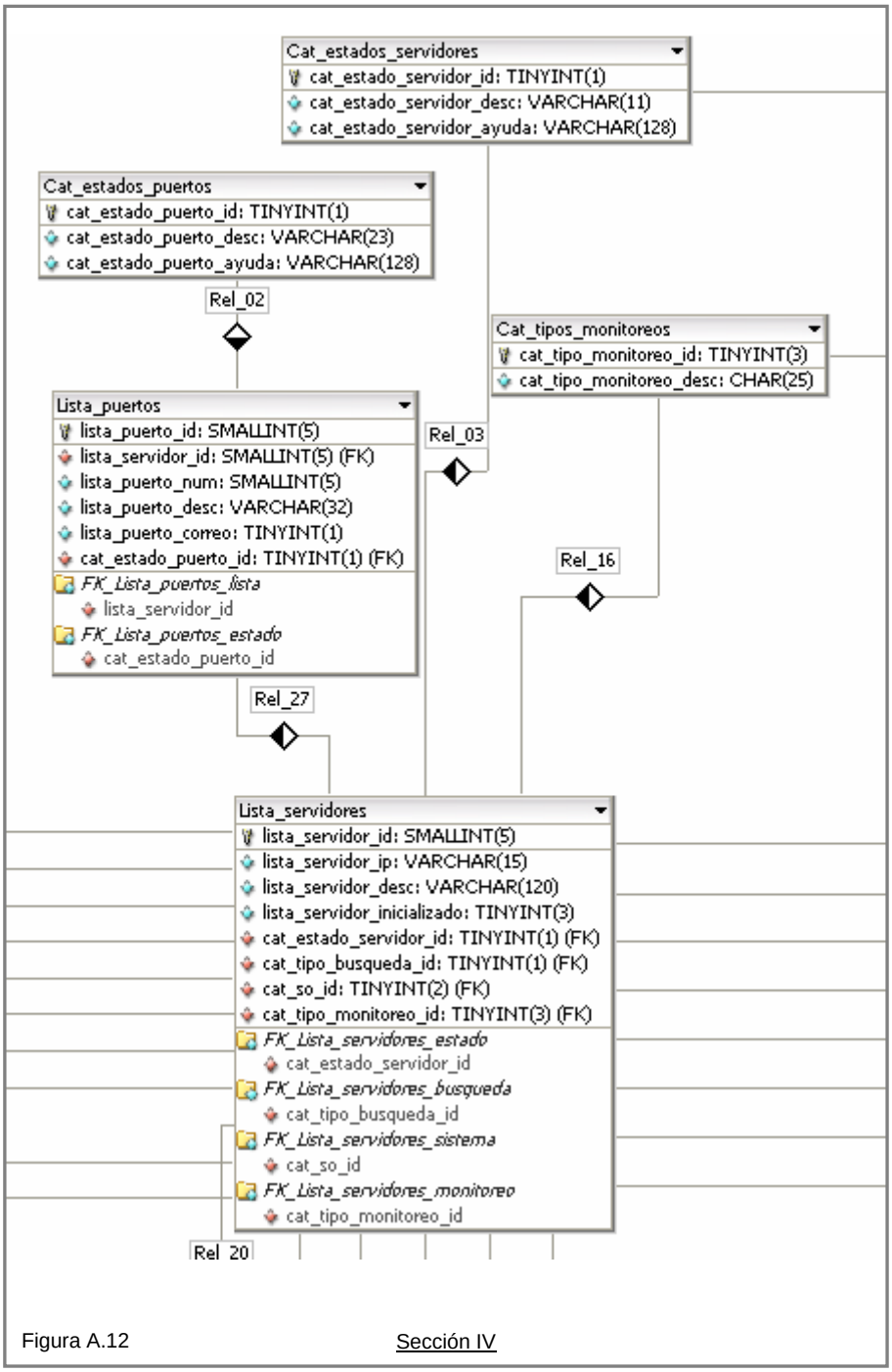


Figura A.12

Sección IV

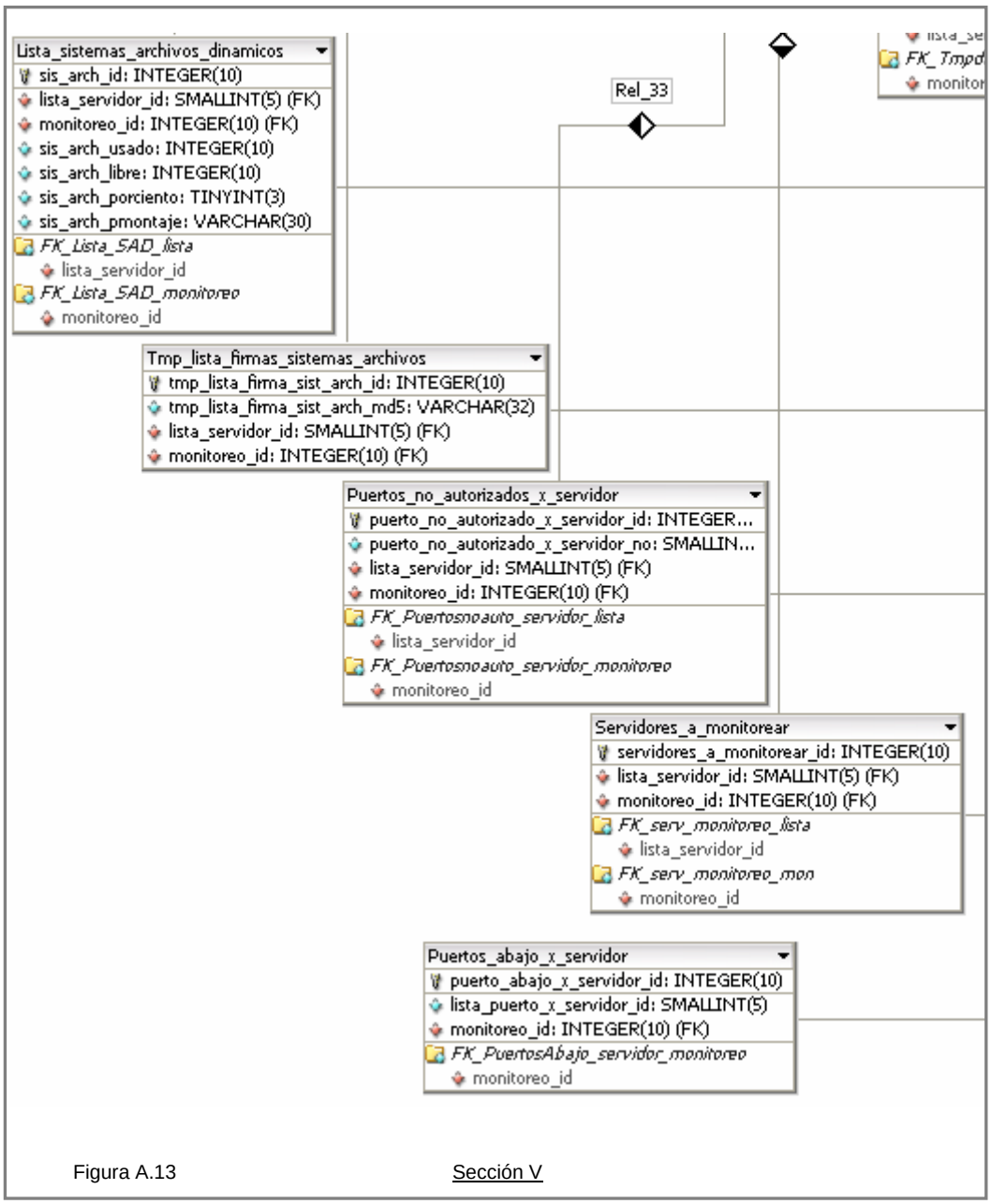


Figura A.13 Sección V

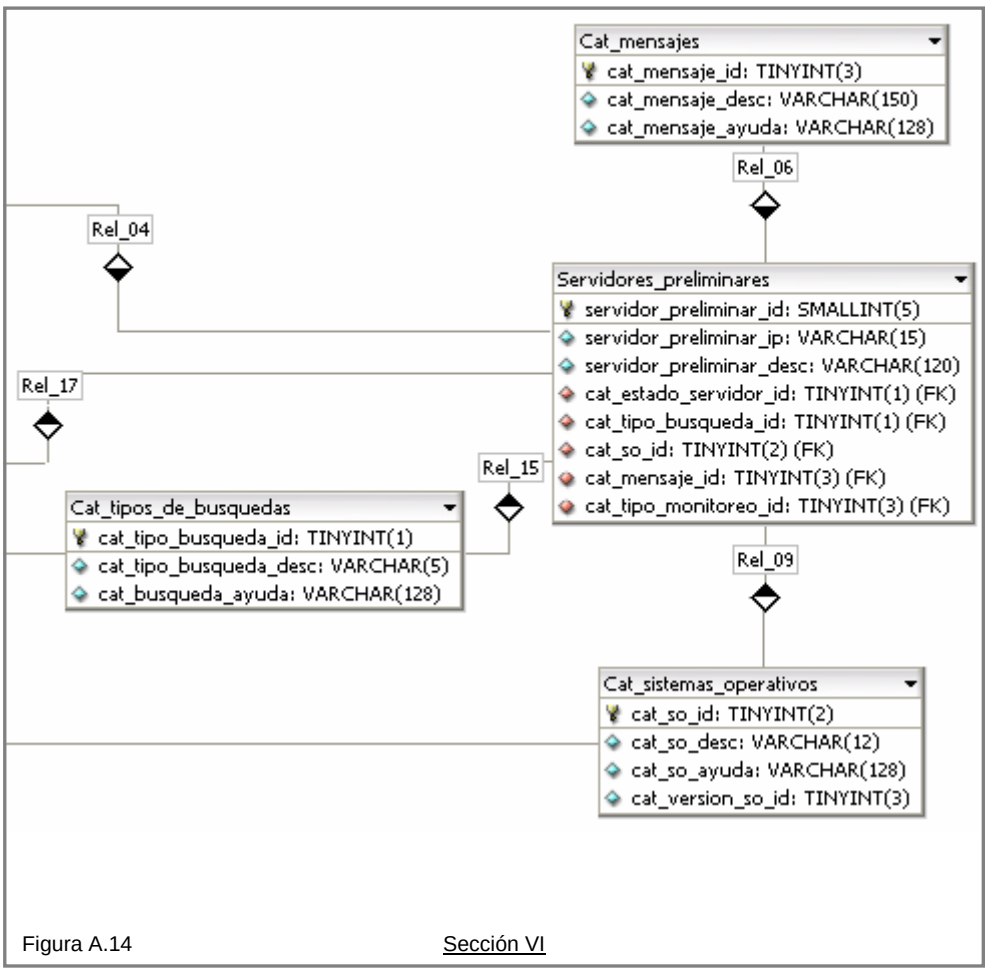


Figura A.14 Sección VI

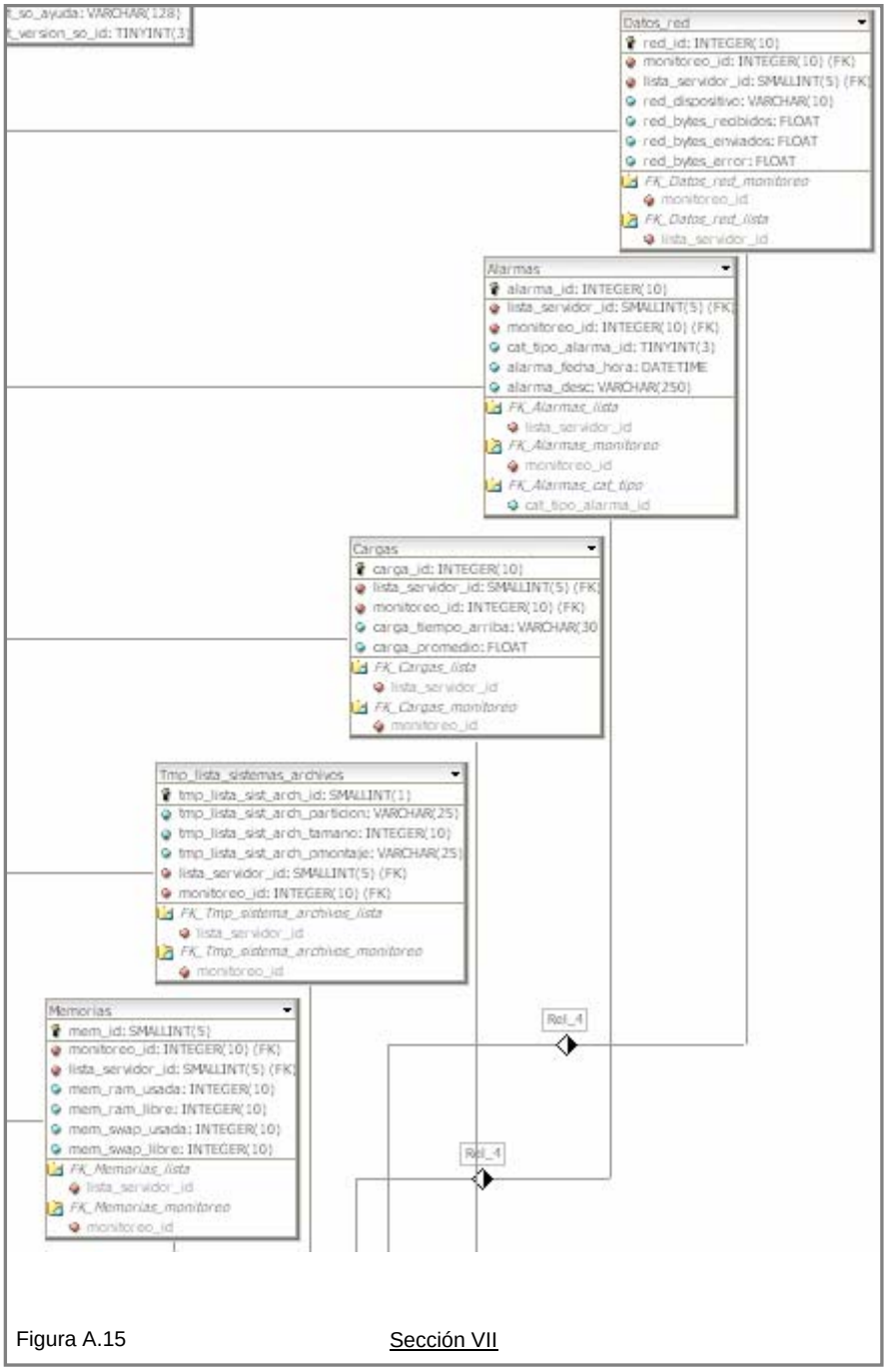


Figura A.15

Sección VII

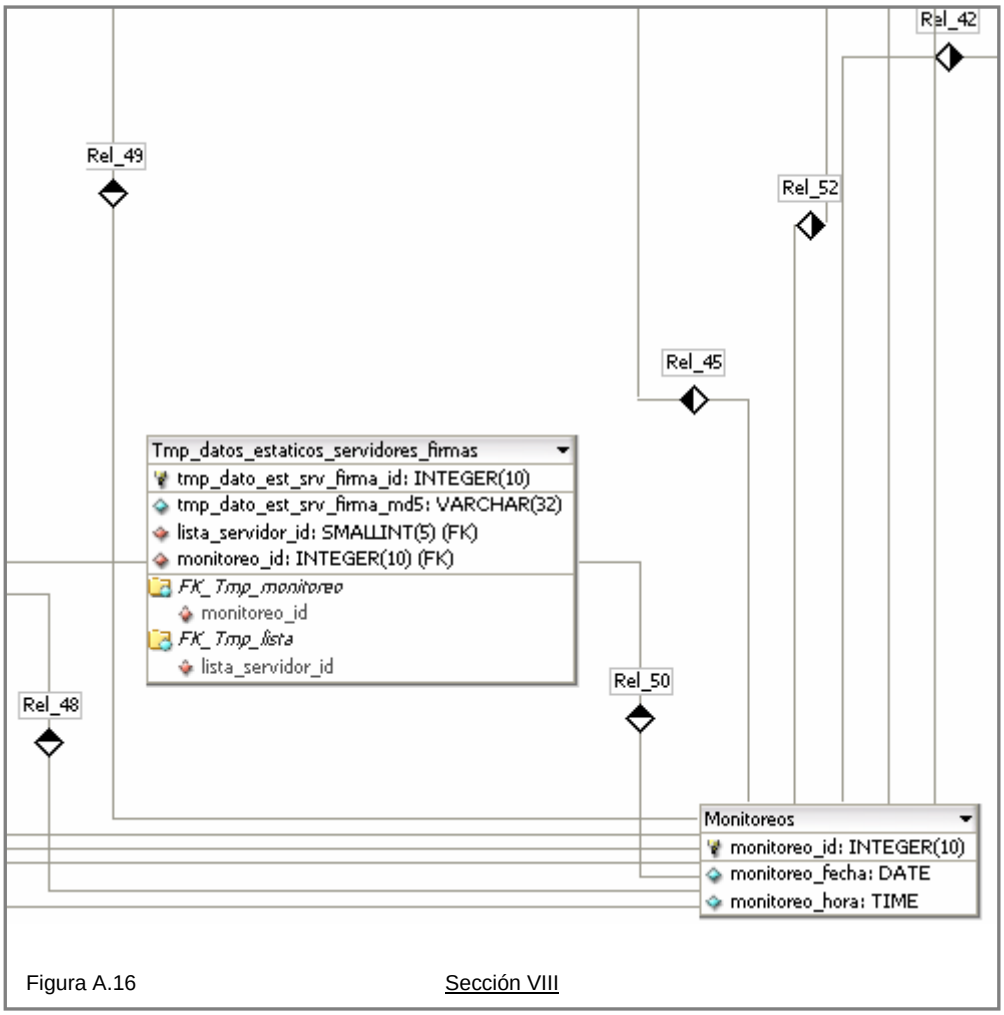


Figura A.16 Sección VIII


# Glosario

**ANSI**
Instituto Nacional Americano de Estándares o American National Standards Institute. Es una organización encargada de estandarizar ciertas tecnologías en EEUU. Es miembro de la ISO.

ANSI es una organización privada sin fines de lucro, que permite la estandarización de productos, servicios, procesos, sistemas y personal en Estados Unidos. Además, ANSI se coordina con estándares internacionales para asegurar que los productos estadounidenses puedan ser usados a nivel mundial.

**AWK**
Es un lenguaje diseñado para procesar datos basados en texto, ya sean archivos o flujos de datos. El nombre AWK deriva de los apellidos de los autores: Alfred Aho, Peter Weinberger y Brian Kernighan.

**Background**
Modalidad de ejecución de procesos en al cual el proceso se disocia de la consola o terminal y el sistema realiza la o las tareas en segundo plano sin bloquear la consola o terminal.

**BSD**
Son las iniciales de Berkeley Software Distribution (en español Distribución de Software de Berkeley) y se utiliza para identificar un sistema operativo derivado del sistema Unix nacido a partir de las aportaciones realizadas a ese sistema por la Universidad de California en Berkeley. Algunos sistemas derivados del BSD son: SunOS, FreeBSD, NetBSD, OpenBSD y Mac OS X.

**CASE**
Software de Ingeniería Asistido por Computo, CASE de sus siglas en ingles (Computer Aided Software Engineering). Estas aplicaciones generan parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

**GNU**
Es un proyecto que inició en 1984 para desarrollar un sistema operativo tipo Unix completamente libre. El kernel o núcleo GNU no fue terminado, así que el kernel utilizado fue el de Linux. La combinación de Linux y GNU es el sistema operativo GNU/Linux.

**GPL (General Public Licence)**
Es una licencia creada por la Free Software Foundation a mediados de los 80, y está orientada principalemente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

La licencia GPL, al ser un documento que cede ciertos derechos al usuario, asume la forma de un contrato, por lo que usualmente se le denomina contrato de licencia o acuerdo de licencia.

**Handshaking**
Es un proceso automatizado de negociación el cual dinámicamente se establecen los parámetros de un canal comunicación establecido entre dos entidades antes de que la comunicación normal sobre el canal inicie. Éste sigue el establecimiento físico del canal y precede a la transferencia normal de información.

**Homófono**
Homófonos son palabras que suenan igual, pero significan algo diferente. Homófono viene del griego homos (igual o semejante) y fonos (sonido).

La palabra en inglés eunuch proviene griego eune que significa cama y ekhein que significa cuidador o guardián. Un eunuch es un hombre castrado.

**ICMP**
Protocolo de Mensajes de Control de Internet (por sus siglas en inglés, Internet Control Message Protocol) es el protocolo de control y notificación de errores del Protocolo de Internet

(IP). Como tal, se usa para enviar mensajes de error, indicando por ejemplo que un servicio determinado no está disponible o que un ruteador o dispositivo no puede ser alcanzado.

**InnoDB**
Es una tecnología de almacenamiento de datos de fuente abierta para MySQL, incluido como formato de tabla estándar en todas las distribuciones de MySQL AB a partir de las versiones 4.0. Su característica principal es que soporta transacciones de tipo ACID y bloqueo de registros e integridad referencial. InnoDB ofrece una fiabilidad y consistencia muy superior a MyISAM, la anterior tecnología de tablas de MySQL.

En octubre de 2005, Oracle Corporation adquirió a la finlandesa Innobase, compañía que desarrollaba InnoDB.

**LAN**
Red de área local (por sus siglas en ingles Local Area Network), es la interconexión de varios dispositivos de red. Su extensión esta limitada físicamente a un edificio o a un entorno de pocos kilómetros. Su aplicación más extendida es la interconexión de computadoras personales y estaciones de trabajo en oficinas, fábricas, etc., para compartir recursos e intercambiar datos y aplicaciones.

**Lenguaje interpretado**
Es aquel en el que las instrucciones se traducen una a una (siendo típicamente unas 10 veces más lentos que los programas compilados). Lenguaje de programación que fue diseñado para ser ejecutado por medio de un intérprete, en contraste con los lenguajes compilado. También se les conoce como lenguajes de script.

**Lenguaje compilado**
Lenguaje de programación que típicamente se implementa mediante un compilador. Esto implica que una vez escrito el programa, éste se traduce a partir de su código fuente por medio de un compilador en un archivo ejecutable.

Los lenguajes compilados son lenguajes de medio o bajo nivel en los que las instrucciones se traducen del lenguaje utilizado a código máquina para un a ejecución rápida.

**Lenguaje de programación multiparadigma**
Es un lenguaje que soporta más de un paradigma de programación. Permite crear programas utilizando más de un estilo de programación.

**Mainframe**
Es una computadora grande, potente y costosa usada principalmente por una gran compañía para el procesamiento de una gran cantidad de datos; por ejemplo, para el procesamiento de transacciones bancarias

**MIT**
Instituto Tecnológico de Massachusetts (del inglés Massachusetts Institute of Technology), es una de las primeras instituciones dedicadas a la docencia y la investigación en los Estados Unidos.

**Multics**
(Multiplexed Information and Computing Service) fue uno de los primeros sistemas operativos de tiempo compartido y tuvo una gran influencia en el desarrollo de los posteriores sistemas operativos.

**Multiusuario**
Se refiere a un concepto de sistemas operativos pero en ocasiones también puede aplicarse a programas de computadora de otro tipo (como por ejemplo aplicaciones de bases de datos). En general se le llama multiusuario al sistema operativo o programa que es capaz de proveer servicios y procesamiento a múltiples usuario simultáneamente (tanto en paralelismo real como simulado).

En contra posición con los sistemas monousuario, que proveen servicio y procesamiento a un solo usuario.

**MyISAM**
Es la tecnología de almacenamiento de datos usada por defecto por el sistema administrador de bases de datos relacionales MySQL. Este tipo de tablas están basadas en el formato ISAM pero con nuevas extensiones.

Cada tabla de tipo MyISAM se guarda en tres archivos cada uno con el nombre de la tabla pero con diferente extensión.

.frm almacena la definición de la tabla
.MYD (MyData) contiene los registros de la tabla
.MYI (MyIndex) contiene los índices de la tabla

Para especificar que deseas usar el tipo de tablas MyISAM, se indica con la opción ENGINE=MYISAM al crear la tabla o modificarla.

La principal característica de este tipo de almacenamiento es la gran velocidad que obtiene en las consultas, ya que no tiene que hacer comprobaciones de la integridad referencial, ni bloquear las tablas para realizar las operaciones por la ausencia de características de atomicidad. Este tipo de tablas está especialmente indicado para sistemas que no tienen un número elevado de inserciones como pueden ser las páginas Web.

**POSIX**
Es el acrónimo de Portable Operating System Interface; la X viene de Unix como señal de identidad de la API. El término POSIX fue sugerido por Richard Stallma en respuesta a la demanda de la IEEE, que buscaba un nombre fácil de recordar. Una traducción aproximada del acrónimo podría ser "Interfaz de Sistema Operativo Portable basado en UNIX".

Familia de estándares de llamada al sistema operativo definidos por el IEEE. Persiguen generalizar las interfases de los sistemas operativos para que una misma aplicación pueda ejecutarse en distintas plataformas.

**RAM**
La memoria de acceso aleatorio, o memoria de acceso directo (en inglés, Random Access Memory), o más conocida como memoria RAM, se compone de uno o más chips y se utiliza como memoria de trabajo para programas y datos. Es un tipo de memoria temporal que pierde sus datos cuando se queda sin energía.

La denominación surgió antiguamente para diferenciarlas de las memorias de acceso secuencial. Debido a que en los comienzos de la computación las memorias principales (o primarias) de las computadoras eran siempre de tipo RAM y las memorias secundarias (o masivas) eran de acceso secuencial (cintas o tarjetas perforadas), es frecuente que se hable de memoria RAM para hacer referencia a la memoria principal de una computadora, pero actualmente la denominación no es demasiado acertada.

**Shebang**
El shebang también llamado hashbang, hashpling o poundbang y se refiere al par de caracteres “#!” que cuando son usados como primeros dos caracteres en la primera línea de un script, causa que los sistemas operativos tipo Unix ejecuten ese script utilizando el interprete especificado por el resto de esa línea.

**Software**
Instrucciones (programas de computadora) que cuando se ejecutan proporcionan la función y el rendimiento deseados. Estructuras de datos que permiten a los programas manipular adecuadamente la información. Conjunto de programas computacionales y procedimientos necesarios para hacer posible la realización de una tarea específica.

**SQL**

El Lenguaje de consulta estructurado (SQL del inglés, Structured Query Language) es un lenguaje de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar información de interés de una base de datos, de una forma sencilla. Es un lenguaje de cuarta generación.

**Swap**
Es un área en el disco duro que temporalmente almacena una imagen de los procesos. Cuando la memoria física demandada no es suficiente, las imágenes de los procesos son regresadas a un área del disco denominada swap en espera de que el sistema libere memoria física para entonces subir la imagen a la memoria física y ser procesada.

**System V**
Abreviado comúnmente SysV y raramente System 5, fue una de las versiones del sistema operativo Unix. Fue desarrollado originalmente por AT&T y lanzado por primera vez en 1983.

**WAM**
Red de Área Amplia (del inglés Wide Area Network), es un tipo de red de computadoras capaz de cubrir distancias desde unos 100 hasta unos 1000 Km, dando el servicio a un país o un continente. Un ejemplo de este tipo de redes sería Internet.


# Bibliografía

## Bibliografía

Ball, Bill and Pitts, David
“Red Hat Unleashed”
Editorial SAMS.

Dale, Dougherty & Robbi, Arnold
“Sed & Awk de O'Reilly"
Editorial O’Reilly segunda edición, 1997.

Frish, Eleen
“Essential System Administration”,
Editorial O’Reilly, Second Edition, 1995.

Kric, Edward V.
“Introducción a la ingeniería y al diseño en la ingeniería”
Editorial Limusa, S.A. de C.V. 2002.

Orwant Jon and Larry Wall, Christiansen
“Perl programming”
Editorial O’Reilly 3ra edición 1990.

Pressman Roger S.
“Ingeniería del software, un enfoque práctico”
Editorial Mc Graw Hill, cuarta edición, 1998.

Simson Garfinkel and Gene Spafford
“Practical UNIX and Internet security”
Editorial O’Reilly 3ra edición 2003.

## Mesografía

**http://www.fi.net.ar/laboratorios/lsi/c-icie99-ingenieriasoftwareeducativo.pdf**
**http://www.angelfire.com/scifi/jzavalar/apuntes/IngSoftware.html**
**http://en.wikipedia.org/wiki/Shell_%28computing%29**
**http://es.wikipedia.org/wiki/Bash**
**http://docs.hp.com/en/B2355-90046/ch02s02.html**
**http://en.wikipedia.org/wiki/History_of_the_Internet**
**http://www.tuobra.unam.mx/publicadas/010815132146-Title.html**
**http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/rmon.htm**
**http://www.securityfocus.com/infocus/1696**
**http://www.faqs.org/rfcs/rfc1157.html**
**http://ceres.ugr.es/~alumnos/qder/indice.html**
**http://plan9.bell-labs.com/who/dmr/chist.html**
**http://www.securityfocus.com/infocus/1696**
**http://www.slac.stanford.edu/comp/net/wan-mon/passive-vs-active.html**
**http://www.nagios.org/**
**http://net-snmp.sourceforge.net/**
**http://www.cacti.net/**
**http://h4ck1t.blogspot.com/2007/06/des-y-triple-des.html**