INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS ESTADO DE MEXICO

PROGRAMA DE GRADUADOS DE INGENIERIA Y CIENCIAS

TESIS

SUPERVISION Y CONTROL DE LOS SERVICIOS BASICOS DE CONECTIVIDAD EN REDES SOPORTADO POR UN SISTEMA AUTOMATICO PARA TOMA DE DECISIONES

QUE PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS DE LA COMPUTACION

CON ESPECIALIDAD EN REDES DE COMPUTADORAS

155785
BIBLIOTECA
21 FEB. 2002

PRESENTA

WENCESLAO GRANDE OLGUIN

ASESOR: DR. EDUARDO J. GARCIA GARCIA

COMITE DE TESIS: DR. LUIS A. TREJO RODRIGUEZ

DR. ROBERTO GOMEZ CARDENAS

Atizapán de Zaragoza, Estado de México, Octubre del 2001.

SUPERVISION Y CONTROL DE LOS SERVICIOS BASICOS DE CONECTIVIDAD EN REDES SOPORTADO POR UN SISTEMA AUTOMATICO PARA TOMA DE DECISIONES

CONTENIDO

Resumen 1
Abstract 2

1. Administración de Redes de Computadoras. 3
   1.1. Introducción. 3
   1.2. Administración con Soporte Experto. 5
   1.3. Desarrollo de la Investigación. 6

2. SNMP, la Infraestructura para la Administración de Redes. 8
   2.1. Resumen Histórico. 8
      2.1.1. Antecedentes. 8
      2.1.2. Los Documentos RFC. 8
      2.1.3. Los RFC de SNMP. 9
      2.1.4. Los RFC de SMI. 10
      2.1.5. Los RFC de las MIB. 11
   2.2. ASN.1, una Notación para Definir Estructuras de Datos. 11
      2.2.1. Áreas de Aplicación. 11
      2.2.2. Conceptos Básicos. 12
      2.2.3. La Norma ASN.1. 13
      2.2.4. Especificación de Protocolos. 14
      2.2.5. Los Módulos ASN.1. 16
      2.2.6. Los Tipos Primitivos de Datos. 19
      2.2.7. Los Tipos Constructores de Datos. 20
      2.2.8. Las Macros. 20
      2.2.9. La Estructura Global de Nombres. 22
      2.2.10. Las Reglas de Codificación. 22
   2.3. SNMP, un Protocolo Simple para la Administración de Redes. 27
      2.3.1. El Modelo Arquitectónico. 27
      2.3.2. Los Elementos de la Arquitectura. 28
      2.3.3. El Protocolo SNMP. 30
      2.3.4. La Transmisión y la Recepción de Mensajes. 31
      2.3.5. PDUs, las Unidades de Datos del Protocolo. 32
   2.4. SMI, la Norma para Describir a los Objetos. 37
      2.4.1. Los Elementos Descriptores. 37
      2.4.2. El Nombre para Identificar. 37
      2.4.3. La Sintaxis para Definir la Estructura. 38
      2.4.4. La Codificación para Representar la Instancia. 40
   2.5. MIB, el Almacén de los Tipos de Datos. 40
      2.5.1. La Especificación Formal de los Tipos de Objetos. 40
      2.5.2. El Agrupamiento de Objetos en las MIBs. 42
      2.5.3. La Actualización de las MIBs. 43

3. Tcl, un Lenguaje Integrador de Sistemas. 45
3.1. El Lenguaje Tcl. 45
3.2. Los Comandos. 48
3.2.1. El Agrupamiento de Argumentos. 48
3.3. Las Expresiones. 48
3.3.1. Las Expresiones Regulares. 50
3.3.2. Los Patrones Globales 52
3.4. Las Listas. 52
3.5. Las Sustituciones. 52
3.6. Los Procedimientos. 53
3.7. Las Variables. 54
3.8. Los Comandos Internos. 54
3.9. Las Funciones Matemáticas Internas. 56
3.10. La Ejecución de Comandos. 57
3.11. Los Resultados de un Comando. 58
3.12. La Instalación de Tcl. 58
3.13. El Aprendizaje de Tcl. 59
3.14. Un Ejemplo de Aplicación de Tcl. 59
3.14.1. El Servidor de Puerto Paralelo. 59
3.14.2. El Cliente. 65

4. El Software SNMP para Administración de Redes. 68
4.1. Scotty, un Software SNMP en Tcl/Tk. 68
4.2. Tnm, un Conjunto de Comandos Textuales. 69
4.3. Los Comandos Tnm::snmp. 70
4.3.1. La Sesión SNMP. 70
4.3.2. Los Comandos SNMP. 71
4.3.3. Los Comandos Generales de Sesión. 72
4.3.4. Los Comandos de las Sesiones 73
4.3.5. La Lista de Variables en SNMP. 74
4.3.6. Los Tipos de Datos en SNMP. 75
4.3.7. Las Operaciones Síncronas ó Bloqueantes. 75
4.3.8. Las Operaciones Asíncronas ó No Bloqueantes. 76
4.3.9. La Configuración de Agentes SNMP. 78
4.4. Los Comandos Tnm::mib. 78
4.5. Ejemplos de Aplicación de Tnm. 80
4.5.1. El Arbol de Identificación de Objetos OID. 80
4.5.2. Los Nodos MIB con Acceso Read-Write. 81

5. La Implementación del Sistema Experto Inicial. 83
5.1. La Arquitectura del Sistema Experto. 83
5.2. La Representación del Conocimiento Experto. 84
5.3. Las Técnicas de Inferencia. 85
5.4. El Lenguaje Prolog. 86
5.5. El Sistema Experto Inicial. 87
5.6. La Prueba Interactiva del Sistema Experto Inicial. 89

6. Administración Automática de Agentes SNMP con Soporte Experto. 91
6.1. El Modelo Agente-Administrador-Experto. 91
6.2. El Sistema de Administración. 92
6.3. El Sistema Experto. 95
6.4. El Interfaz de Operación. 97
6.5. La Prueba del Modelo. 101

7. Conclusiones. 105

8. Referencias. 107

Resumen.

Las redes de computadoras se han convertido en un elemento primordial para el manejo de la información. Mantenerlas operando con un alto nivel de calidad es una prioridad que requiere una administración apropiada. Este reto se ha atendido principalmente con alternativas basadas en el Protocolo Simple para Administración de Redes (SNMP), dentro del conjunto de protocolos TCP/IP. Las implementaciones de este protocolo ofrecen una infraestructura operativa para la administración de redes. SNMP ha tenido aceptación en operaciones de supervisión pero las operaciones de control se han limitado, debido a la fragilidad del mecanismo de seguridad de la versión más popular, SNMPv1. La última versión, SNMPv3, atiende el aspecto de seguridad, además de mantener la compatibilidad con versiones anteriores.

Otros retos dentro de la administración de redes se derivan de su crecimiento, de su complejidad y de su importancia. Uno de ellos es el manejar de la mejor forma, en el menor tiempo y al menor costo, las fallas operativas de la red, especialmente ante eventos que degradan considerablemente su desempeño y donde se cuenta con un procedimiento de recuperación, previamente probado por los expertos. Si un sistema experto puede modelar el razonamiento humano con programas de cómputo y es posible codificar la experiencia en este dominio del conocimiento, entonces parece factible proporcionar soporte experto a la administración de redes, disminuyendo la necesidad de requerir expertos humanos en el lugar y el momento en que se necesiten, lo que podría significar un esfuerzo humano y económico prohibitivo.

El objetivo de la presente investigación es mostrar la viabilidad de proporcionar soporte experto a los sistemas de administración de redes de manera automática. Se propone como solución un sistema basado en un Modelo Agente-Administrador-Experto que integra técnicas de administración de redes con técnicas de sistemas expertos. El administrador supervisa a un agente para obtener información operativa, con tal información consulta al sistema experto y con el resultado de la consulta envía acciones de control al agente.

La investigación presenta el estudio realizado del Protocolo SNMP, de la norma ASN.1 usada para definir la sintaxis abstracta de los objetos y la sintaxis de transferencia de los mensajes, de la Estructura de la Información Administrativa SMI que define el nombre y la sintaxis de los objetos, y del almacén virtual denominado Base de Información Administrativa (MIB-2). El programa administrador se desarrolló en Lenguaje Tcl para facilitar la integración de las aplicaciones. Se usó Scotty, una implementación gratuita del Protocolo SNMP desarrollado en Lenguaje Tcl/Tk para plataformas Unix y Windows. El sistema experto se desarrolló en Lenguaje Prolog y se opera de manera automática a través de un Interfaz de Operación desarrollado en Lenguaje C.

Los resultados de la prueba del modelo muestran la viabilidad de proporcionar soporte experto automático a los sistemas de administración de redes.

Palabras Clave: Administración de Redes, ASN.1, MIB-2, Modelo Agente-Administrador-Experto, Prolog, Scotty, Sistemas Expertos, SMI, SNMP, Tcl/Tk.

Abstract.

The computer networks have become a fundamental element for the information handling. To maintain operating them with a high quality level is a priority that requires an appropriate administration. This challenge has been attended mainly with alternatives based on the Simple Network Management Protocol (SNMP), within the TCP/IP protocol set. The implementations of this protocol offer an operative infrastructure for the network management. SNMP has been accepted in supervision operations but the control operations have been limited, due to the SNMPv1 security mechanism fragility, its most popular version. SNMPv3, its last version, care of the security aspect, besides to maintain the compatibility with previous versions.

Other challenges within the network management are derived from their growth, its complexity and its importance. One of them is to handle in the best way, less time and less cost, the operative faults of the network, specially with events that degrade considerably their performance and where it is counted on a recovery procedure, previously proven by the experts. If an expert system can model the human reasoning with computer programs and is possible to codify the experience in this domain of the knowledge, then it seems feasible to provide expert support to the network management, diminishing the necessity to require human experts in the place and the moment at which they are needed, which could mean a prohibitive human and economic effort.

The objective of the present research is to show the viability to provide expert support to the networks management systems in automatic way. It is proposed a system based solution on a Agent-Administrator-Expert Model, which integrates techniques of network management with techniques of expert systems. The administrator supervises to an agent to obtain operative information, with such information it consults to the expert system and with the result of the consultation sends control actions to the agent.

The research presents the study of SNMP Protocol, of ASN.1 norm used to define the objects abstract syntax and the messages transfer syntax, of the Structure of Management Information (SMI) that defines the name and the syntax of the objects, and of Management Information Base (MIB-2) the objects virtual warehouse. The administrator program was developed in Tcl Language to facilitate the integration of the applications. Scotty was used, a free implementation of SNMP Protocol, developed in Tcl/Tk Language for Unix and Windows plataforms. The expert system was developed in Prolog Language and it operates in automatic way through an Operation Interface developed in C Language.

The test results of model, show the viability to provide automatic expert support to the network management systems.

Keywords: Network Management, ASN.1, MIB-2, Agent-Administrator-Expert Model, Prolog, Scotty, Expert systems, SMI, SNMP, Tcl/Tk.

1. Administración de Redes de Computadoras.

1.1. Introducción.

Los sistemas de procesamiento distribuido y las redes de computadoras han incrementado su número e importancia en el mundo actual. Las organizaciones requieren cada vez más del apoyo de las redes y de sus recursos asociados, con un nivel de desempeño que les permita desarrollar sus funciones de una manera aceptable. El número de actividades que dependen del funcionamiento de las redes ha crecido a un grado en que, mantenerlas operativas la mayor parte del tiempo es una alta prioridad. El usuario ha incrementado su dependencia de tales servicios y se espera, por tanto, ofrecerle un servicio de calidad que soporte sus requerimientos.

Ofrecer un servicio de calidad requiere de una administración apropiada. La International Standard Organization (ISO), ha establecido un marco de referencia administrativo para las redes de computadoras, dividiendo sus tareas en 5 áreas [1][2]:

- Administración de la Configuración,
- Administración del Desempeño,
- Administración de las Fallas,
- Administración de la Seguridad y
- Administración Contable.

Estas funciones administrativas requieren de una infraestructura para poder ser implementadas. En la redes que usan el conjunto de protocolos Transmission Control Protocol/Internet Protocol (TCP/IP), la infraestructura que permite la administración de las redes de computadoras esta basada principalmente en el Protocolo Simple para la Administración de Redes (SNMP, Simple Network Management Protocol). Este protocolo nació como una solución a corto plazo para atender los problemas de la administración de redes [3].

El protocolo SNMP funciona bajo una variante del paradigma cliente/servidor, denominada agente/administrador. Los agentes operan en el dispositivo administrable y funcionan como servidores que actualizan y proporcionan información de sus objetos según se las solicite el administrador. El administrador funciona como cliente, realizando dos tipos de operación: supervisar al agente, solicitando información de sus objetos y controlar al agente, solicitando modificaciones en los datos de sus objetos. Estas operaciones son del tipo solicitud-respuesta. Una tercera operación, iniciada por el agente, permite enviar notificaciones de manera asíncrona al administrador, reportando la ocurrencia de ciertos eventos predeterminados.

Probablemente las principales ventajas de este protocolo sean su popularidad y su capacidad de expansión. La mayoría de los elementos administrables que operan en la actualidad soportan SNMP versión 1 y las principales compañías fabricantes de equipos para redes, han desarrollado bases de datos privadas para administrar sus equipos.

SNMP es un protocolo que se desarrolló inicialmente para atender necesidades administrativas relativamente simples, por lo que la versión 1 no satisface ciertas necesidades administrativas actuales. Entre estas desventajas se cuenta su baja eficiencia y la que es considerada su principal desventaja, su mínima seguridad. El mecanismo de seguridad de SNMPv1 esta basado en un código conocido como *comunidad*, el cual identifica a los miembros que pertenecen a un grupo determinado de agentes y administradores. Los administradores miembros de tal comunidad tienen derechos predeterminados para solicitar ciertas operaciones de los agentes. Este código se maneja en texto claro en los mensajes entre administradores y agentes, lo que lo hace muy vulnerable ante ataques a su seguridad. La versión 2 de SNMP es más segura pero no tuvo gran aceptación dada la complejidad de su implementación [4]. La versión SNMPv3, cuyas implementaciones se están probando, pretende resolver finalmente el problema de la inseguridad.

La seguridad en la administración de redes es particularmente sensible, en especial cuando se tiene el propósito no solo de supervisar la red sino de controlarla. La administración de redes de computadoras usando como infraestructura a SNMPv1, se ha enfocado principalmente a la supervisión de las redes, más que al control de la misma, como consecuencia de la popularidad de la versión 1 y de su frágil mecanismo de seguridad.

Además de los problemas de seguridad, que se esperan resolver con la versión SNMPv3, las cada vez más exigentes necesidades administrativas, plantean nuevas necesidades que requieren de estrategias y herramientas apropiadas. Los actuales retos operativos de los servicios de red, deben apoyarse con facilidades adicionales en la administración de redes, a fin de poder atender los problemas que se han generado con el crecimiento de su número, de su complejidad y de la dependencia en sus servicios.

Los administradores de redes requieren de herramientas que permitan manejar en el menor tiempo y de la mejor forma, las fallas operativas de la red. Estas herramientas deberían inclusive, contar con la posibilidad de automatizar el control de la red, particularmente ante eventos que degradan considerablemente su desempeño y donde se cuenta con un procedimiento de recuperación plenamente probado.

La diversidad de los servicios de red requeridos, su volumen, su mayor complejidad, la necesidad de soportar sistemas abiertos, el costo de tales servicios, la necesidad de respuestas rápidas a los problemas y la calidad de servicio requerida, ponen a la administración de redes en un plano relevante dentro de la problemática actual de las tecnologías de la información.

La administración de redes de computadoras ha enfrentado retos considerables. Centralizar su operación ha sido una de las primeras medidas que se han tomado. Centralizada la administración, el siguiente paso ha sido contar con expertos en el área a fin de poder atender sin límite de tiempo, ni de horario, ni de grado de dificultad, los diversos problemas que se presentan en la operación diaria de las redes.

El costo y la dificultad de administrar la red bajo estos requerimientos es alto. Las administraciones de tamaño mediano y pequeño, donde se administran solo algunos cientos de computadoras, probablemente tendrán dificultades económicas para lograrlo. Aún las

administraciones de gran tamaño y que cuentan con recursos para realizarlo, probablemente ponderarán la carga económica necesaria contra la calidad de servicio deseada.

En cualquier caso, sería conveniente contar con una alternativa para administrar redes de computadoras que cuenten con apoyo experto de manera continua y a un costo asequible. Esta alternativa debería proporcionar respuestas efectivas y automatizadas a los problemas más comunes de la red. Si contar con la experiencia en el dominio de conocimiento, en el momento mismo de suceder el problema, es una necesidad para mejorar la operación de la red, se podría implementar una solución considerando el soporte de sistemas expertos.

1.2. Administración con Soporte Experto.

Para poder lograr, respuestas efectivas y automáticas en los centros administrativos de redes, esta investigación propone dotar a los sistemas de administración de red con técnicas del área de sistemas expertos.

Los sistemas expertos tratan de modelar el razonamiento humano con programas de cómputo. Se considera que el razonamiento humano se realiza por medio de la inferencia. Existen dos estrategias básicas de inferencia: la primera deduce información nueva a partir de información conocida y la segunda establece primero una hipótesis y luego pretende validarla. El primer tipo de inferencia utiliza fundamentalmente la regla de inferencia conocida como modus ponens y se le conoce como inferencia con encadenamiento hacia delante o guiada por datos. La segunda, utiliza el procedimiento de unificación y resolución para tratar de determinar si la meta propuesta u hipótesis es verdadera y se le conoce como inferencia con encadenamiento hacia atrás o guiada por metas [5][6].

El desarrollo de los sistemas expertos se ha realizado en diferentes lenguajes, pero Prolog tiene características que lo hacen apropiado para su implementación. Una de tales características es que cuenta con un motor de inferencia, el cual funciona con encadenamiento hacia atrás o guiado por metas. El motor de inferencia realiza su función con hechos almacenados en la base de hechos y con conocimientos contenidos en la base de conocimiento, para derivar conclusiones.

Cuando el sistema de administración consulte al sistema experto, le enviará los hechos. El sistema experto derivará una conclusión y la enviará como respuesta. Por ejemplo, si el motor de inferencia del sistema experto recibe como un hecho el que el límite de fallas se haya excedido para un cierto objeto, debería concluir y enviar como respuesta que es necesario una acción correctiva, la cual podría ser, desde un reinicio del sistema, hasta la actualización de alguna variable específica del objeto, que permita resolver el problema en cuestión.

El conocimiento en el dominio, proporcionado por los expertos, se encuentra codificado en la base de conocimiento. Este tipo de conocimiento es heurístico. El conocimiento heurístico es la traducción del conocimiento profundo de un problema basado en leyes fundamentales a reglas operativas simples obtenidas con la experiencia, que facilitan considerablemente la solución del problema.

Dadas las características de los sistemas expertos, su base de conocimiento es un elemento independiente del motor de inferencia. La base de conocimiento se puede enriquecer constantemente, incluyendo nuevos conocimientos que los expertos vayan deduciendo con el tiempo. Inicialmente se podrían automatizar los procesos para resolver las fallas cuyas soluciones sean bien conocidas. Las fallas que aún no se consideren en este rango, se orientarían para su atención en la forma acostumbrada.

Los sistemas expertos, al codificar la experiencia de los expertos en la base de conocimiento, permiten aprovecharla sin las limitantes asociadas a los expertos humanos. Los sistemas de administración de red podrían contar con los servicios de un motor de inferencia, que les proporcione las respuestas efectivas, rápidas y de la calidad que proporcionaría un experto humano, utilizando una base de conocimiento que permita usar el conocimiento experto en línea.

El sistema que se propone en esta investigación, integra técnicas de administración de redes de computadoras con técnicas de sistemas expertos. Las características de los sistemas expertos permiten que al aumentar el conocimiento experto en redes de computadoras, éste se agregue a su base de conocimiento y se pueda usar por el motor de inferencia para derivar conclusiones. Con esta facilidad en línea, se puede disponer del conocimiento que los expertos han obtenido a través de su experiencia laboral sin la necesidad de contar con su presencia física. Al contar con conocimiento experto en una base de conocimiento, el costo de usar tal conocimiento disminuye considerablemente en comparación con el costo de tener un experto humano atendiendo a la red, además de liberar al experto humano de problemas repetitivos de relativa complejidad.

Esta investigación pretende mostrar como un sistema de administración de redes puede tomar decisiones relacionadas con los agentes que administrade mnaera automática, por medio de un motor de inferencia que opera una base de conocimiento la cual puede codificar la experiencia obtenida en administración de redes.

1.3. Desarrollo de la Investigación.

El Capítulo 2, "SNMP, la Infraestructura para la Administración de Redes", presenta al Protocolo Simple para la Administración de Redes (SNMP, Simple Network Management Protocol). Se describe su desarrollo histórico, su arquitectura, sus operaciones, los objetos de su Base de Información Administrativa conocida como MIB-2 (Management Information Base II) y la forma de describirlos según la norma Estructura de la Información Administrativa (SMI, Structure of Management Information), incluyendo información sobre su notación y las reglas de codificación para su transmisión.

El Capítulo 3, "Tcl, un Lenguaje Integrador de Sistemas", describe al lenguaje que se usó como elemento integrador de las diferentes aplicaciones que se usaron. Se explican las características de Tcl, incluyendo su metodología para ejecutar comandos: agrupar, sustituir y ejecutar. Como un ejemplo de las facilidades de Tcl para la intercomunicación de procesos, se

muestra una aplicación que permite controlar el puerto paralelo de una computadora remota a través de una red TCP/IP.

El Capítulo 4, "El Software SNMP para Administración de Redes", presenta al sistema administrador de redes que se usó en la investigación. Existen sistemas de administración desarrollados en lenguajes como C++, Java, Perl y Tcl principalmente. Algunos sistemas ofrecen un entorno que facilita su programación, cuentan con bibliotecas API cuyas funciones SNMP pueden ser llamadas para realizar las operaciones correspondientes. Considerando que el objetivo fundamental de la investigación es mostrar la posibilidad de que un sistema de administración de redes cuente con el apoyo de una máquina de inferencia para automatizar las decisiones que controlarán a la red se seleccionó un sistema que, por una parte, satisface en alto grado las funciones que actualmente realizan la mayoría de los sistemas administradores SNMP y que, por otra, permite una fácil interacción con otros sistemas.

El sistema administrador seleccionado fue Scotty, un producto gratuito desarrollado en Tcl/Tk que puede operar bajo Unix o Windows. Entre sus características estan la representación gráfica de los nodos de la red, el descubrimiento automático de nodos, la visualización de la información de sus objetos administrables, el soporte a la Base de Información Administrativa MIB-2 y a algunas otras MIB de compañías particulares.

El Capítulo 5, "La Implementación del Sistema Experto Inicial", presenta la arquitectura del sistema propuesto. Esta arquitectura consta de una interfaz de operación en lenguaje C y de un sistema experto incrustado que se desarrolló en lenguaje Prolog. Se presenta una introducción al motor de inferencia de Prolog y se muestra la base de conocimiento utilizada.

El Capítulo 6, "Administración Automática de Agentes SNMP con Soporte Experto", es el capítulo integrador. En esta capítulo se muestra la esencia de la propuesta: la posibilidad de que un sistema administrador de redes pueda recibir apoyo automático de un sistema experto que tiene codificada la experiencia de los expertos en el área. Se muestran los resultados de una administración automática, útil en la etapa operativa.

El Capítulo 7, "Conclusiones", incluye los comentarios relacionados con las experiencias obtenidas, las dificultades encontradas y las sugerencias de desarrollo posterior.

El Capítulo 8, "Referencias", lista las fuentes de información que se usaron en la investigación.

2. SNMP, la Infraestructura para la Administración de Redes.

2.1. Resumen Histórico.

2.1.1. Antecedentes.

En la década de los 80's, se incrementó el interés por administrar las redes TCP/IP. Dada la necesidad de generar protocolos que atendieran estas necesidades, el Internet Activities Board (IAB), encargado del desarrollo técnico del conjunto de protocolos de Internet, emitió en abril de 1988 el documento "IAB Recommendations for the Development of Internet Network Management Standards" (RFC 1052) desarrollado por Vincent Cerf, para recomendar la generación de dos protocolos: uno que atendería las necesidades a corto plazo y otro que incluiría más facilidades y operaría a largo plazo.

Para lograr el protocolo a corto plazo, se modificó un protocolo denominado Simple Gateway Monitoring Protocol (SGMP) y se creó el protocolo Simple Network Management Protocol (SNMP).

Para lograr el protocolo a largo plazo, se desarrollaría un protocolo de administración de redes dentro del Modelo OSI denominado Common Management Information Protocol (CMIP), que incluiría facilidades más amplias.

Dentro de IAB, el grupo Internet Engineering Task Force (IETF) es el responsable de las tareas a corto plazo, mientras que el Internet Research Task Force (IRTF) es el responsable de las tareas a largo plazo. Como resultado de las recomendaciones del RFC 1052, se crearon tres grupos de trabajo dentro de IETF:

- El Grupo de Trabajo MIB.
  Se formó para desarrollar un marco de referencia común para la administración de redes TCP/IP.
- El Grupo de Trabajo SNMP Extensions.
  Se formó para adaptar el protocolo SGMP con las experiencias obtenidas y con los resultados del Grupo de Trabajo MIB.
- El Grupo de Trabajo NETMAN.
  Se formó para explorar el marco administrativo de la red en OSI y desarrollar CMIP sobre TCP/IP (CMOT).

2.1.2. Los Documentos RFC.

Los grupos de trabajo antes mencionados así como otras entidades de investigación sobre Internet, pueden generar un conjunto de documentos como resultado de su trabajo, denominados *Internet-Drafts*. Cuando se publican los *Internet-Drafts* se pueden categorizar como *Standard Track*, *Experimental* o *Informational*. Los documentos *Standard Track*

intentan convertirse en una norma y entran a este proceso en el estado de *Proposed-Draft*, pueden avanzar a *Draft* y finalmente convertirse a *Standard (full)*. Algunos documentos no alcanzan su normalización o son sustituídos por otros y terminan siendo calificados como *Obsolete* o *Historic*, respectivamente.

Cada documento que se publica tiene un número que lo identifica denominado *Request for Comments (RFCs)*. Estos documentos se pueden consultar en IETF [7]. El desarrollo histórico de SNMP se refleja oficialmente, de manera similar a otros protocolos del conjunto de protocolos TCP/IP, en sus RFCs. Es de observar que continuamente se generan nuevas especificaciones y que el estado de las existentes se altera, por lo que se debe consultar al IETF para obtener información actualizada.

2.1.3. Los RFC de SNMP.

El antecedente principal de SNMP se conoce como Simple Gateway Monitoring Protocol (SGMP, RFC 1028), data de noviembre de 1987 y fue desarrollado por Davin, Case, Fedor y Schoffstall. Al año siguiente, los mismos autores desarrollan la definición de SNMP y en 1990 publican su primera versión conocida como SNMPv1, la cual ha mantenido su estado *full* por más de una década. El éxito de esta versión ha permitido el desarrollo inicial de la infraestructura necesaria para la administración de redes, pero sus limitantes y las crecientes necesidades en esta área, han requerido de propuestas más avanzadas. La siguiente lista muestra un resumen del desarrollo de las normas RFC relacionadas con SNMP [8]:

- SNMPv1 (*full*).
  RFC 1157 (mayo de 1990), reemplaza a RFC 1067 y RFC 1098. Primera versión del protocolo SNMP desarrollado por Davin, Case, Fedor y Schoffstall. La seguridad está basada en cadenas de caracteres denominadas *comunidades (communities)*.
- SNMPsec (*historic*).
  RFC 1351, RFC 1352 y RFC 1353. Esta versión incluyó características fuertes de seguridad, basadas en *contextos de operaciones restringidas (parties)*.
- SNMPv2p (*historic*).
  RFC 1441, RFC 1445, RFC 1446, RFC 1448 y RFC 1449. Actualizó a SNMPv1 y a SMIv1 en operaciones del protocolo, tipos de datos y seguridad basada en *contextos de operaciones restringidas (parties)* de SNMPsec. Se le conoce también como SNMPv2 classic.
- SNMPv2c (*experimental*).
  RFC 1901, RFC 1905 y RFC 1906. Actualiza las operaciones del protocolo y los tipos de datos de SNMPv2p. Usa el modelo de seguridad basado en *comunidades* de SNMPv1. Se le conoce también como SNMPv2 basada en comunidades.
- SNMPv2u (*experimental*).
  RFC 1905, RFC 1906, RFC 1909 y RFC 1910. Usa las operaciones del protocolo y el tipo de datos de SNMPv2c con un modelo de seguridad basado en *usuarios*.

- SNMPv2* (*experimental*).
  Los RFC que definen esta versión no se publicaron. Combinó lo mejor de SNMPv2p y SNMPv2u. Se conoce también como SNMPv2star.
- SNMPv3 (*draft standard*).
  RFC 2570 (abril de 1999) a RFC 2576 (marzo del 2000). Combina un modelo de seguridad basado en *usuarios* derivado de SNMPv2u y SNMPv2* y las operaciones del protocolo y los tipos de datos de SNMPv2p. Soporta *apoderados (proxies)*. Cada entidad SNMP está formada por un Motor y por Aplicaciones [9].

  El Motor de una entidad SNMPv3 consta de 4 partes:
  - Despachador de Mensajes.
  - Procesador de Mensajes.
    Soporta SNMPv1, SNMPv2c, SNMPv3 y otros protocolos.
  - Servicios de Seguridad.
    Autentificación y privacidad de mensajes. Soporta el modelo de seguridad basado en *comunidades*, en *usuarios* y *otros modelos*.
  - Control de Acceso.
    Soporta el modelo de control de acceso basado en *vistas* y *otros modelos*.

  Entre las Aplicaciones de una entidad SNMPv3 se consideran:
  - Generadores de Comandos.
  - Respondedores de Comandos.
  - Originadores de Notificaciones.
  - Receptores de Notificaciones.
  - Reenviadores de Apoderados.

SNMPv3 es la última versión del protocolo SNMP y se espera que resuelva la necesidad de operaciones seguras.

2.1.4. Los RFC de SMI.

El protocolo SNMP administra objetos. Cada tipo de objeto tiene un nombre, una sintaxis y una codificación:

- Nombre.
  El nombre de un tipo de objeto permite su identificación y se representa con un identificador de objeto.
- Sintaxis.
  La sintaxis define la estructura de datos abstracta o tipo de dato de un objeto.
- Codificación.
  La codificación es la forma de instanciar el valor de un objeto considerando su sintaxis y determinadas reglas de codificación.

El nombre y la sintaxis de los objetos administrables se especifican en el documento Estructura de la Información Administrativa (Structure of Management Information, SMI). Se tienen las siguientes versiones de SMI:

- SMIv1 (full).
  RFC 1155 (mayo de 1990) desarrollada por Rose y McCloghrie. El documento RFC 1215 especifica como definir traps y complementa al RFC 1155.
- SMIv2 (draft).
  RFC 2578 (abril de 1999), RFC 2579 y RFC 2580. Desarrollada por McCloghrie, Perkins y Schoenwaelder, amplía SMIv1.

2.1.5. Los RFC de las MIB.

Además de especificar el protocolo y la forma de definir el nombre y la sintaxis de los objetos administrables, es necesario definir el almacén virtual de información denominado Base de Información Administrativa (Management Information Base, MIB), por medio del cual se accesa a los objetos administrables. En la MIB se encuentra definido un conjunto de tipos de objetos integrados en grupos. En estos grupos se han incluído aquellos objetos que se consideraron esenciales para administrar las fallas y la configuración, y que su manejo, en caso de operaciones erróneas, no causaría daños mayores a la operatividad de los sistemas. Las especificaciones MIB son las siguientes:

- MIB (historic).
  RFC 1156 (mayo de 1990), desarrollada por McCloghrie y Rose. Define la primera versión de MIB.
- MIB-2 (full).
  RFC 1213 (marzo de 1991), desarrollada por McCloghrie y Rose y actualizada en RFC 2011, 2012 y 2013. Compatible con la primera versión de MIB.

2.2. ASN.1, una Notación para Definir Estructuras de Datos.

2.2.1. Áreas de Aplicación.

El rápido avance de los sistemas distribuidos requiere intercambios de información entre computadoras, que cada vez son más complejos. La especificación de los formatos de los mensajes requiere ser normalizado por medio de protocolos de aplicación a fin de asegurar que la comunicación sea posible entre los diferentes dispositivos.

Para especificar a los protocolos de aplicación, se ha seleccionado al lenguaje ASN.1 (Abstract Syntax Notation 1) creado por Larmouth, Steedman y White. ASN.1 permite una

implementación precisa, eficiente y sin errores. Su estudio permitirá comprender diferentes normas, particularmente las normas completas (Full Standards) de SNMPv1:

- RFC 1155, Structure of Management Information (SMI),
- RFC 1157, Simple Network Management Protocol (SNMP) y
- RFC 1213, Management Information Base II (MIB-2).

ASN.1, permite describir:

- Estructuras Abstractas de Datos.
  Permite describir la notación para definir las estructuras de datos de los mensajes de manera abstracta, es decir independientemente de su implementación o de cualquier otra restricción.
- Reglas de Codificación.
  Permite describir las reglas asociadas de codificación que determinan los patrones precisos de bits, los cuales representarán valores de tales estructuras de datos cuando se transfieran a través de una red de computadoras, independientemente de la plataforma y del lenguaje.
- Compiladores ASN.1.
  Permite describir las herramientas de software conocidas como Compiladores ASN.1 que convierten las definiciones en ASN.1 a definiciones de estructuras de datos en un lenguaje de computación en particular. Estas estructuras se usan en el desarrollo de las aplicaciones que requieren el intercambio de mensajes.

2.2.2. Conceptos Básicos.

Inicialmente, en la notación de los tipos de datos se usó el término virtualización para indicar la separación de los tipos de datos virtuales intercambiados por un protocolo, de las estructuras de datos reales de una implementación en particular. Actualmente se usan los términos sintaxis abstracta y sintaxis de transferencia, pero los conceptos básicamente son los mismos:

- La Sintaxis Abstracta.
  Permite describir a los tipos de datos independientemente de su representación física o alguna otra restricción.

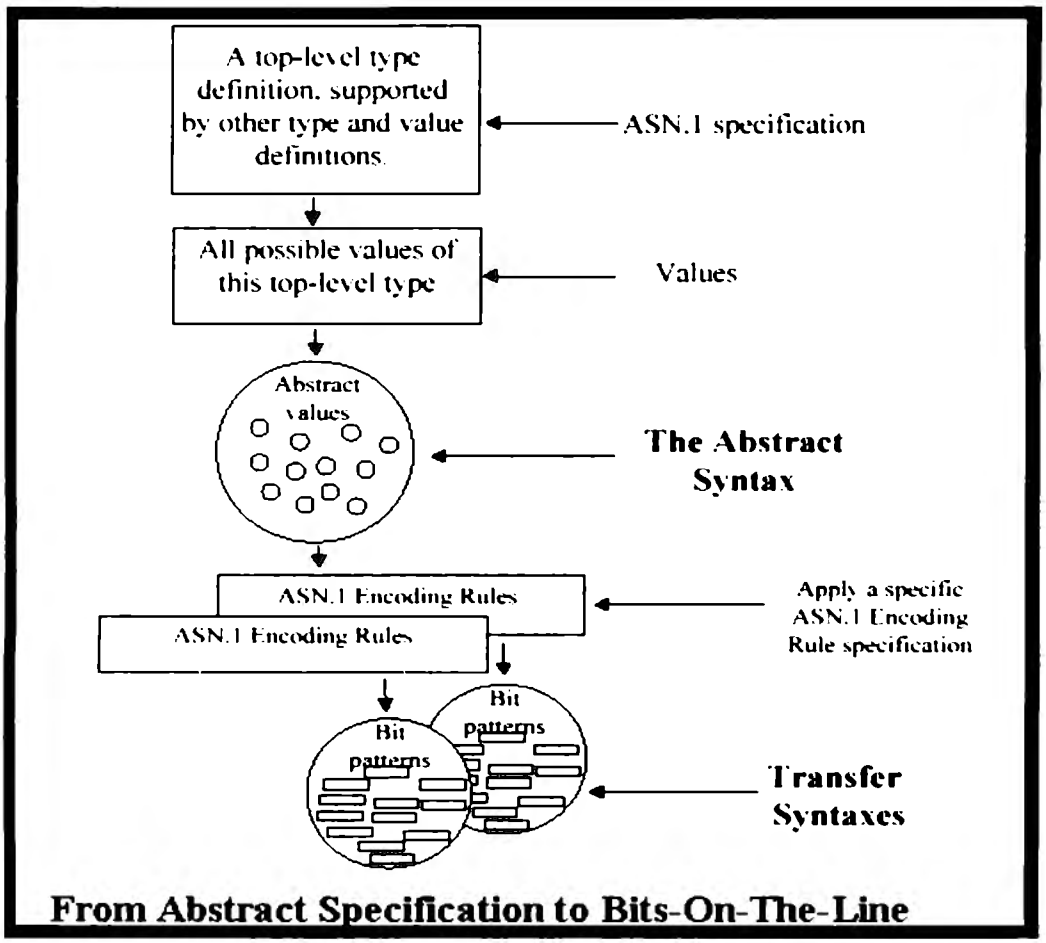
- La Sintaxis de Transferencia.
  Permite describir la representación concreta de los tipos de datos o instancias, que pueden ser transmitidos por la red usando flujos de octetos o de bits entre el transmisor y el receptor.

Los conflictos iniciales que se atendían en la representación abstracta estuvieron enfocados a evitar errores en el ordenamiento de los bits y de los octetos en las estructuras de datos, defininiendo por ejemplo, si se transmitía bajo la estructura Litle Endian o Big Endian.

Posteriormente los tipos de datos se hicieron más complejos y se necesitó un lenguaje formal para obtener definiciones precisas. ASN.1 es el lenguaje más común para describir los tipos de datos de los mensajes transmitidos por los protocolos de telecomunicaciones.

El mecanismo para producir las representaciones concretas (Sintaxis de Transferencia), son las Reglas de Codificación. Una de las más comunes son las Reglas Básicas de Codificación (Basic Encoding Rules, BER) definidas para ASN.1.

La siguiente figura tomada de esta referencia [10], muestra la relación entre la Sintaxis Abstracta y la Sintaxis de Transferencia.



2.2.3. La Norma ASN.1.

ASN.1 fue estandarizado en 1984 como X.409 por CCITT (International Telegraph and Telephone Consultative Committe), ahora ITU-T (International Telecommunication Union – Telecommunication Standardization Sector). ISO adoptó esta notación y dividió la recomendacion en dos documentos: la Sintaxis Abstracta (ASN.1) y las Reglas de Codificación (BER). En 1987, ISO publicó estos documentos como ISO 8824 e ISO 8825 [11].

En 1988, ISO junto con IEC (International Electrotechnical Commission) formaron un comité técnico denominado ISO/IEC JTC 1, que ahora está a cargo de la norma ASN.1. En 1989 CCITT publicó en su Libro Azul las recomendaciones X.208 y X.209, una revisión de ASN.1, que proveía extensiones, resultado del trabajo en JTC 1.

Para la versión siguiente llamada ASN.1:1994, la Sintaxis Abstracta, norma ISO 8824, se dividió en cuatro partes:

- ISO 8824-1 ó ITU-T X.680: Especificación de la Notación Básica,
- ISO 8824-2 ó ITU-T X.681: Especificación del Objeto de la Información,
- ISO 8824-3 ó ITU-T X.682: Especificación de las Restricciones y
- ISO 8824-4 ó ITU-T X.683: Parametrización de las Especificaciones de ASN.1.

En cuanto a las Reglas de Codificación, la norma ISO 8825 se dividió en dos partes:

- ISO 8825-1 ó ITU-T X.690: Especificación de las Reglas Básicas de Codificación (BER), de las Reglas Canónicas de Codificación (CER) y de las Reglas Distinguidas de Codificación (DER) e
- ISO 8825-2 ó ITU-T X.691: Especificación de las Reglas Empacadas de Codificación (PER).

Existe una nueva versión denominada ASN.1:1997, la cual es una edición corregida de ASN.1:1994.

2.2.4. Especificación de Protocolos.

Se puede considerar a un protocolo de comunicaciones como un conjunto definido de mensajes, cadenas de caracteres o de bits los cuales tienen un significado definido conocido como semántica del mensaje y un conjunto de reglas, que permiten decidir cuando se debe enviar un mensaje en particular.

En las redes TCP/IP un protocolo es parte de una pila de protocolos. Los protocolos trabajan juntos para integrar un mensaje completo que será emitido por el transmisor. Una especificación para un determinado nivel de la pila de protocolos, determina la forma y el significado de las partes externas (encabezado y cola) del mensaje de tal nivel. Este mensaje generalmente proporciona un servicio de transporte a un mensaje de un nivel superior que se transporta en su parte interna.

Los mensajes de estos protocolos pueden ser especificados de dos maneras diferentes :

- Especificaciones Basadas en Caracteres.
  El protocolo se define como una serie de líneas de texto codificadas en ASCII.
- Especificaciones Basadas en Bits.
  El protocolo se define como una cadena de bits o como una cadena de grupos de ocho bits (octetos).

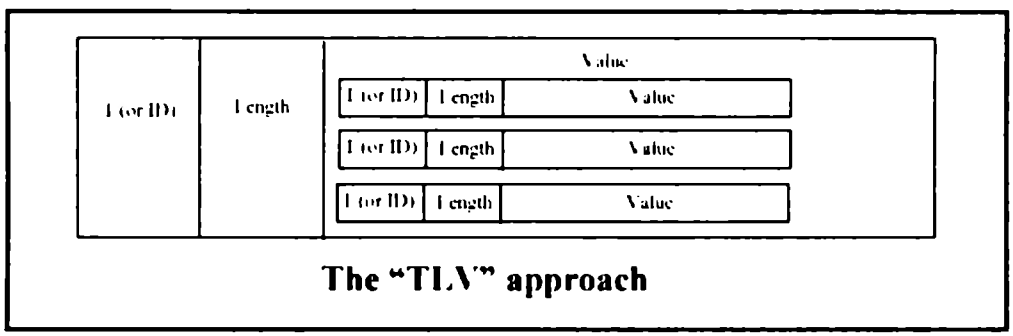
Cuando se usan especificaciones basadas en caracteres, el enfoque no es una estrucutra de datos de propósito general sino un conjunto de líneas de texto que se pueden considerar como comandos o instrucciones. Un ejemplo es el protocolo HTTP. Otro ejemplo es la notación BNF (Bachus-Naur Form) la cual es poderosa para definir estructuras sintácticas pero tiene algunas limitantes: no determina como delimitar elementos de longitud variable ni como determinar el número de iteraciones. Bajo este enfoque, los campos de longitud se reemplazan por caracteres reservados que sirven para delimitar la longitud de un campo.

Un ejemplo de especificaciones basadas en bits, además de ASN.1, es el que se usa en el Lenguaje para Definición de Interfaces (Interface Definition Language, IDL). En IDL el modelo es un conjunto de objetos interactuando a través de una red por medio de interfaces predefinidas, las cuales permiten invocar a los métodos de los objetos. Una implementación IDL es la arquitectura CORBA (Common Object Request Broker Architecture). CORBA IDL está enfocado a un modelo de invocación y respuesta mientras que ASN.1 está enfocado a especificaciones más generales y flexibles.

Uno de los conceptos más importantes que se han usado en los mensajes para definir la información asociada a cada elemento, es conocido como *TLV*: *Tipo*, *Longitud* y *Valor*. Bajo el concepto TLV todos los elementos de información de un mensaje se codifican con:

- Un Campo de Tipo (T).
  Identifica la naturaleza del elemento.
- Un Campo de longitud (L).
  Delimita la longitud del valor.
- Un Campo de Valor (V).
  Contiene la información del elemento.

El Campo de Valor V, puede contener elementos básicos de valor o elementos completos TLV, lo que permite una estructura de árbol de prácticamente cualquier profundidad [10]



Cuando se especifica un protocolo, gran parte de la información se puede expresar como campos de longitud fija y de presencia obligatoria. Estas especificaciones se han definido tradicionalmente por medio de diagramas.

Este enfoque ha funcionado bien, exceptuando quizás, la falta de claridad cuando se tenía que definir cual de los octetos era el más significativo y cual debía transmitirse primero.

Pero los protocolos también requieren de la posibilidad de mensajes de longitud variable con parámetros que puedan ser opcionales. La especificación contendría entonces, algunos campos obligatorios de longitud fija, seguidos por un campo paramétrico el cual estaría integrado por una serie de parámetros, cada uno con sus valores TLV.

Otro requerimiento de los protocolos sería la posibilidad de manejar grupos opcionales de parámetros y de manejar un número variable de repeticiones de un determinado parámetro. Este enfoque es la base de la codificación BER.

Para BER el mensaje completo está integrado por un identificador de tipo (T), un campo de longitud (L) y el valor o cuerpo del mensaje (V). En estas tripletas TLV, el campo de valor V puede contener otra tripleta TLV o un valor básico. Este mecanismo permite definir grupos de tipos y repeticiones de tipos.

Con el desarrollo de las codificaciones CER y PER, los mensajes se pueden integrar de manera más eficiente, al evitarse información superflua y compactarse su codificación.

2.2.5. Los Módulos ASN.1.

Las especificaciones ASN.1 se realizan por módulos. Un módulo ASN.1 contiene:

- Una Sección de Encabezado,
- Una Sección de Definición de Etiquetas y Extensibilidad,
- Una Instrucción BEGIN,
- Una Sección de Exportación e Importación,
- Una Sección de Especificación de Tipos, Valores y Macros y
- Una Instrucción END.

En la descripción del módulo, se siguen ciertas convenciones:

- Comentarios: se inician con la secuencia "- -" y terminan al final de la línea.
- Espacios: se pueden intercambiar con finales de línea.
- Palabras Reservadas: se escriben en MAYÚSCULAS.
- Tipos de Datos Internos de ASN.1: se escriben en MAYÚSCULAS
- Nombres de Tipos de Datos: empiezan con Mayúscula.
- Nombres de Elementos, Alternativas o Enumeraciones: empiezan con minúscula.

El Encabezado se localiza al principio e inicia con el Nombre del Módulo, cuya letra inicial debe estar en mayúscula.

Enseguida se encuentra el Identificador del Módulo el cual no es obligatorio pero recomendable a fin de definir la identificación del módulo dentro del espacio global de nombres.

La siguiente línea contiene la palabra reservada DEFINITIONS donde se define la forma de manejar el etiquetado y la extensibilidad de los elementos.

Para el manejo del etiquetado (números de identificación: [0], [1], etc.), se tenían dos posibilidades: un etiquetado implícito (IMPLICIT TAGS) y un etiquetado explícto (EXPLICIT TAGS). Actualmente se usa un etiquetado automático (AUTOMATIC TAGS) el cual agrega automáticamente las etiquetas correspondientes a todos los elementos de cada SEQUENCE o SET y a cada alternativa de CHOICE inciando con la etiqueta "[0]".

La extensibilidad permite ampliar en versiones posteriores, una estructura de datos, manteniendo la compatibilidad con la versión original. Para indicar el punto de inserción de los nuevos elementos, se tienen dos opciones: marcadores explícitos y marcadores implícitos. Si se requieren marcadores explícitos se indica EXTENSIBILITY IMPLIED, si se requieren marcadores explícitos no se indica nada.

Con marcadores explícitos, el punto de inserción de los nuevos elementos debe indicarse con tres puntos denominados elipses (marcador de extensión: ...). Si la elipse está textualmente presente, se considera, si no lo está, se ignora. Con marcadores implícitos, todas las especificaciones de tipo que no contengan un marcador de extensión donde se permita indicarlo, se agregarán al final de la construcción.

El encabezado termina con la palabra reservada BEGIN e inicia la parte donde se indican las definiciones de tipo, de valores y de macros que se pueden exportar (EXPORTS) a otros módulos y las que se pueden importar (IMPORTS FROM) de otros módulos.

Los elementos que se van a exportar se indican después de la palabra EXPORTS separados por comas si es necesario y se finaliza con punto y coma. Si se omite la palabra EXPORTS, significa que se exporta todo. Si se indica EXPORTS solamente, no se exporta nada.

Para importar, se indica la palabra IMPORTS, el elemento o los elementos separados por comas a importar de un mismo módulo, la palabra FROM y el nombre del módulo con su identificación global entre llaves.

Si se requiere importar elementos de otro módulo, se escribe el elemento o elementos a importar, nuevamente la palabra FROM y el módulo donde se encuentra junto con su identificación. El proceso se repite por cada elemento importado. Al final se termina con punto y coma.

Si la identificación de los módulos de los que se importa incluyen referencias a tipos identificadores de objetos (OBJECT IDENTIFIER), estas definiciones se deben incluir enseguida.

La parte siguiente del módulo es la parte medular y contiene las definiciones de tipo, de valor y de macros. Esta parte inicia generalmente indicando el nombre del tipo de más alto nivel, el cual se reconoce por que le da un nombre a la sintaxis abstracta. Después del nombre y su tipo, se define su sintaxis abstracta y se indica su identificación global.

La especificación anterior desencadena la necesidad de especificar tipos de datos y valores hasta llegar a los tipos de datos primitivos de ASN.1. Esta parte del módulo incluye a cada tipo y a cada valor que se haya referenciado previamente.

El módulo termina con la palabra reservada END.

Un ejemplo de un Módulo ASN.1 es el siguiente:

```
RFC1157-SNMP DEFINITIONS ::=
   BEGIN
      IMPORTS ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks
              FROM RFC1155-SMI;

      -- top-level message
      Message ::= SEQUENCE {version   INTEGER {version-1(0)},
                            community OCTET STRING,
                            data      ANY
                           }
      -- protocol data units
      PDUs ::= CHOICE {get-request        GetRequest-PDU,
                       get-next-request   GetNextRequest-PDU,
                       get-response       GetResponse-PDU,
                       set-request        SetRequest-PDU,
                       trap               Trap-PDU
                      }
      -- PDUs
      GetRequest-PDU     ::=  [0]  IMPLICIT PDU
      GetNextRequest-PDU ::=  [1]  IMPLICIT PDU
      GetResponse-PDU    ::=  [2]  IMPLICIT PDU
      SetRequest-PDU     ::=  [3]  IMPLICIT PDU

      PDU ::= SEQUENCE {request-id        INTEGER,
                        error-status      INTEGER {noError(0),
                                                   tooBig(1),
                                                   noSuchName(2),
                                                   badValue(3),
                                                   readOnly(4),
                                                   genErr(5)
                                                  },
                        error-index       INTEGER,
                        variable-bindings VarBindList
                       }
      Trap-PDU ::=
      [4] IMPLICIT SEQUENCE {enterprise        OBJECT IDENTIFIER,
                             agent-addr        NetworkAddress,
                             generic-trap      INTEGER {coldStart(0),
                                                        warmStart(1),
                                                        linkDown(2),
                                                        linkUp(3),
                                                        authenticationFailure(4),
                                                        egpNeighborLoss(5),
                                                        enterpriseSpecific(6)
                                                       },
                             specific-trap     INTEGER,
                             time-stamp        TimeTicks,
                             variable-bindings VarBindList
                            }
      -- variable bindings
      VarBind ::= SEQUENCE {name  ObjectName,
                            value ObjectSyntax
                           }
      VarBindList ::=  SEQUENCE OF VarBind
   END
```

2.2.6. Los Tipos Primitivos de Datos.

ASN.1 tiene un número predefinido de tipos primitivos o básicos, de estructuras de datos.

- El tipo primitivo BOOLEAN consta de los valores abstractos falso y verdadero y la notación de su valor se indica con las palabras "FALSE" y "TRUE".

- El tipo primitivo INTEGER prácticamente no tiene límites en sus valores. Puede incluir una lista de valores distinguidos cualesquiera y/o un rango (subtipo), una especificación de excepción después de la palabra DEFAULT y un marcador de extensión (...).

- El tipo primitivo ENUMERATED consta solamente de los valores correspondientes a los nombres listados. Si no se incluyen los valores numéricos, la enumeración inicia en 0. Puede tener un marcador de extensión (...).

- El tipo primitivo REAL existe en base 10 y en base 2. La notación del valor es una lista de enteros separados por comas que indican la mantisa, la base y el exponente. Se pueden asignar los valores PLUS-INFINITY y MINUS-INFINITY para manejar valores muy grandes.

- El tipo primitivo BIT STRING puede indicar su valor en forma binaria agregando la letra B a su valor entre apóstrofos o puede indicar su valor en forma hexadecimal, en múltiplos de cuatro bits, agregando la letra H a su valor. Las cadenas de bits se usan generalmente para especificar la versión con bits nombrados. Cuando se usan sin bits nombrados, generalmente son parte de una estructura compleja para transportar información encriptada. Excepto para usarse en la versión y en la encriptación, no se recomienda su uso, dado que se considera sería mejor seleccionar otro tipo.

- El tipo primitivo OCTET STRING es el tipo preferido para transportar cualquier tipo de información y su valor se puede indicar en binario o en hexadecimal. No se recomienda ya que generalmente existen otros tipos que pueden transportar la información de manera específica.

- El tipo primitivo NULL se usa normalmente donde se necesita un tipo para definir un elemento, pero no hay información para el valor. Permite llenar una posición.

- Los tipos cadena de caracteres son subconjuntos de caracteres normalizados. Entre estos se encuentran PrintableString, VisibleString y UTF8String. Se recomienda usar UTF8String, el cual implica cualquier carácter en cualquier lenguaje.

- El tipo primitivo OBJECT IDENTIFIER se usa para identificar de manera precisa y global un objeto dentro del espacio de nombres normalizado sin considerar la semántica del objeto (puede ser un documento de normas, un módulo ASN.1, un interfaz, etc.). El espacio de nombres tiene una estructura tipo árbol y se define en SMI (RFC 1155).

- El tipo primitivo ObjectDescriptor (en mayúsculas y minúsculas) permite describir textualmente un objeto.

- El tipo primitivo UTCTime (Coordinated Universal Time) es un tipo fecha/hora que maneja solo dos dígitos para el año. Algunos valores de UTCTime son yymmddhhmmZ y yymmddhhmmssZ. La Z significa Hora del Meridiano de Greenwich.

- El tipo primitivo GeneralizedTime maneja los años en cuatro dígitos. Algunos valores de GeneralizedTime son yyyymmddhhmmZ y yyyymmddhhmmssZ. La Z significa Hora del Meridiano de Greenwich.

2.2.7. Los Tipos Constructores de Datos.

Los tipos básicos permiten construir tipos compuestos de estructuras de datos. Entre los tipos compuestos o constructores, los más usados son:

- Secuencias de elementos (SEQUENCE),
- Conjuntos de elementos (SET) ,
- Selección entre elementos (CHOICE),
- Listas, secuencias repetidas un número arbitrario de veces (SEQUENCE OF) y
- Macros que facilitan la definición de estructuras de datos (MACRO).

Si una estructura se califica como tipo SEQUENCE, significa que es una secuencia de campos TLV cuyo orden indica la secuencia en que serán transmitidos. Un campo tipo SET es similar a SEQUENCE excepto que el conunto de sus elementos se transmiten en un orden que no requiere ser determinado por su definición, el orden queda al arbitrio de los codificadores de los transmisores. Un tipo CHOICE permite seleccionar uno entre varios elementos TLV. Si el campo se califica como SEQUENCE OF, significa que su contenido puede repetirse un número arbitrario de veces, incluyendo cero veces, tal como funcionan las listas.

2.2.8. Las Macros.

Las macros permiten incrementar las capacidades de definición de las estructuras de datos. A diferencia de las macros en los lenguajes de programación donde se realizan substituciones textuales, las macros en ASN.1 redefinen las reglas gramaticales del lenguaje. La definición de una macro es la siguiente:

```
nombre-de-la-macro MACRO      ::=
BEGIN
      TYPE NOTATION           ::= tipo-de-sintaxis
      VALUE NOTATION          ::= valor-de-sintaxis
      sintaxis-de-apoyo
END
```

155785
BIBLIOTECA

Una vez definida la macro, ésta se puede asignar como un tipo de dato a un nombre de un elemento. El tipo de sintaxis define las reglas gramaticales y su valor de sintaxis el valor asociado a cada instancia. La sintaxis de apoyo proporciona cualquier información adicional que se requiera. Por ejemplo, la siguiente definición de la macro OBJECT-TYPE se usa en el RFC 1155 Structured Management Information:

```
OBJECT-TYPE MACRO ::=
BEGIN
    TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
                      "ACCESS" Access
                      "STATUS" Status
    VALUE NOTATION ::= value (VALUE ObjectName)

    Access ::= "read-only"
             | "read-write"
             | "write-only"
             | "not-accessible"
    Status ::= "mandatory"
             | "optional"
             | "obsolete"
END
```

El módulo debe incluir las definiciones de ObjectSyntax y ObjectName nombradas en la macro:

```
ObjectSyntax ::= CHOICE {number     INTEGER,
                         string     OCTET STRING,
                         object     OBJECT IDENTIFIER,
                         empty      NULL
                        }
ObjectName   ::= OBJECT IDENTIFIER
```

Como ejemplo, para definir una instancia de la macro OBJECT-TYPE que se denomine sysDescr, se indicaría:

```
sysDescr    OBJECT_TYPE             -- instancia y nombre de la macro
                                    -- tipo de sintaxis:
            SYNTAX OCTET STRING     -- SYNTAX e instancia de ObjectSyntax
            ACCESS read-only        -- ACCESS e instancia de Access
            STATUS mandatory        -- STATUS e instancia de Status
                                    -- valor de sintaxis:
            ::= {system 1}          -- ::= e instancia de ObjectName
```

Dada la posibilidad de que cada módulo defina sus propios macros, implementar un Compilador ASN.1 que las maneje ha sido difícil. Actualmente las macros han sido substituídas por facilidades orientadas a objetos, sin embargo las macros se han usado en las especificaciones previas a 1994, incluyendo las normas para SNMPv1.

2.2.9. La Estructura Global de Nombres.

El espacio global de nombres está asociado al tipo primitivo OBJECT IDENTIFIER y se usa ampliamente en los protocolos especificados en ASN.1. Este tipo de estructura de datos permite identificar diferentes objetos: módulos ASN.1, objetos administrables, sintaxis abstracta, sintaxis de transferencia y otros objetos de información.

El espacio global de nombres permite identificar de manera unívoca un objeto y compartir tal identificación entre los usuarios. La infraestructura para identificar objetos está formada por un árbol computacional. El árbol se extiende partiendo de una raíz única, y cada nodo es asignado a una autoridad que lo administra.

La parte inicial del espacio global de nombres es la siguiente:

```
root
   (0) itu-t o ccitt
         (0) recommendation
         (1) question
         (2) administration
         (3) network-operator
   (1) iso
         (0) standard
         (1) registration-authority
         (2) member-body
         (3) identified-organization
               (6) dod
                     (1) internet
   (2) joint-iso-itu-t o joint-iso-ccitt
```

2.2.10. Las Reglas de Codificación.

Una vez que ASN.1 nos ha servido para definir las estructuras de los datos, estas especificaciones deben permitir producir las estructuras correspondientes a un lenguaje de programación en particular, usando un Compilador ASN.1. Por ejemplo, para el Lenguaje C, se generarán archivos tipo .h y tipo .c. En otros casos, se generarán los archivos necesarios para producir bibliotecas estáticas lib o dinámicas dll. Estos archivos se incluirán con el código de aplicación para generar el ejecutable final.

El código de aplicación escribirá en las estructuras de datos del lenguaje en particular y cuando la información esté completa, llamará a la rutina de codificación proporcionada por el Compilador ASN.1, para producir una sintaxis de transferencia, es decir, una codificación completa del valor de ese tipo de estructura.

De manera similar pero inversa, se realiza la decodificación [12].

Resumiendo, el módulo ASN.1 se alimenta al Compilador ASN.1 y produce dos salidas principales: las definiciones de las estructuras de datos (archivo .h para el Lenguaje C) y el

código fuente (archivo .c para el Lenguaje C) con las rutinas que codifican y decodifican las estructuras de datos a la sintaxis de transferencia y viceversa.

Este proceso de codificación/decodificación de la sintaxis abstracta a la sintaxis de transferencia se realiza en base a las Reglas de Codificación. Las Reglas de Codificación se pueden considerar parte del Nivel de Presentación del Modelo OSI.

Cuando se definen especificaciones por medio de caracteres, se pueden producir mensajes con información innecesaria o redundante. Este tipo de especificaciones tienden al desuso.

Las especificaciones actuales se enfocan a bits, agrupadas en octetos, como ASN.1.

Con el enfoque de especificaciones basadas en bits, la información a ser enviada en un mensaje se trata como un conjunto de valores paramétricos, donde cada uno tiene una codificación TLV: un parámetro de identificación (T) normalmente fijo de uno o varios octetos, un parámetro de longitud (L) en uno, dos o tres octetos y una codificación del valor (V) como una secuencia de codificación TLV de octetos.

Con este enfoque todos los campos tienen un número entero de octeros y el campo de longitud cuenta octetos, no bits.

Este mecanismo permite enviar a los parámetros en cualquier orden, declarar que algunos parámetros son opcionales, permite manejar parámetros de longitud variable y tiene la posibilidad de manejar la extensibilidad. Este concepto se amplió para incluir el concepto de grupos de parámetros donde se codifica un identificador de grupo, una longitud de grupo y una serie de codificaciones TLV para los parámetros dentro del grupo. Los grupos pueden aparecer en cualquier orden, pueden ser opcionales, manejar longitudes variables y extensibilidad.

Esto nos lleva a dos niveles de TLV: el nivel de grupo y el nivel de parámetro. Este enfoque anidado de TLV fue la base de las Reglas BER.

Para lograr codificaciones más compactas, se desarrollaron otros tipos de reglas de codificación conocidas como CER (Canonical Encoding Rules), DER (Distinguished Encoding Rules) y PER (Packed Encoding Rules). BER permite que el codificador maneje las opciones, DER define a las opciones y PER las elimina. Aún cuando la complejidad aumenta, los mensajes se compactan reduciendo los requerimientos de ancho de banda.

Las especificaciones SNMPv1, se desarrollaron bajo las Reglas de Codificación BER.

Para ASN.1 el orden de transmisión de los octetos se especifica de izquierda (más significativo) a derecha (menos significativo). El orden en que se transmiten serialmente los bits dentro de cada octeto o aún si se transmiten en paralelo, no se determina en ASN.1 sino por los Protocolos de Transporte. El orden de los bits dentro de cada octeto se considera del más significativo (MSB) a la izquierda, al menos significativo (LSB) a la derecha.

La codificación TLV consta de las siguientes partes:

- Tipo (T).

  - Clase

    Se codifica en los 2 primeros bits de la siguiente manera:

    - Universal: Código 00.

      La identificación se realiza a nivel global.

    - Application Wide: Código 01.

      La identificación se realiza a nivel aplicación, es decir es única dentro de un módulo ASN.1.

    - Context-Specific: Código 10.

      La identificación solo es aplicable en la definición de un tipo de dato.

    - Private: Código 11.

      La identificación se realiza a nivel de una organización.

  - Primitivo/Constructor

    Se codifica en el siguiente bit con 0 o 1 respectivamente.

  - Etiqueta

    Se codifica iniciando en los últimos 5 bits del primer octeto:

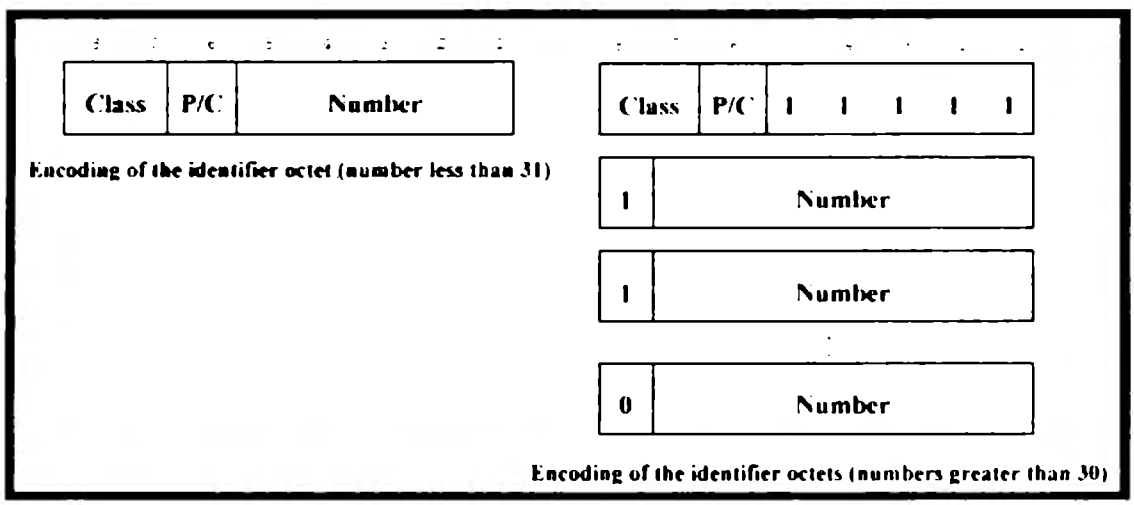
    - Etiquetas de 0 a 30 (00000 a 11110)

      Se codifican en el primer octeto.

    - Etiquetas de 31 en adelante

      El primer octeto se codifica como 11111.

      Las etiquetas iguales o superiores a 31 se codifican en los siguientes octetos considerando los 7 bits menos significativos de cada octeto de manera encadenada. El bit mas significativo tendrá un 1 si la concatencación requiere un octeto adicional y tendrá un 0 en caso de ser el último octeto en la concatenación, por ejemplo: 00 0 11111, 1xxxxxxx, 1xxxxxxx, 0xxxxxxx. En este ejemplo, las x's se concatenan para codificar la etiqueta. Veáse la siguiente figura:

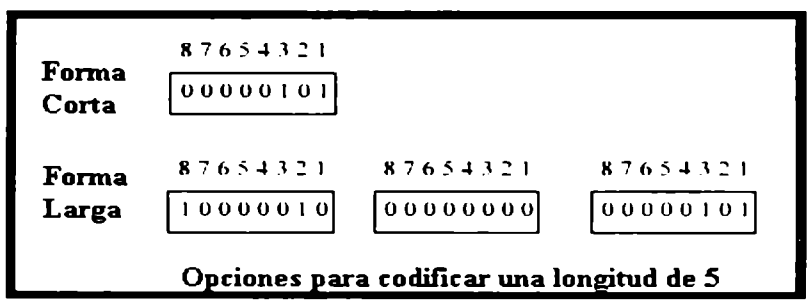


La siguiente tabla muestra algunas codificaciones del Tipo:

```
Class P/C Tag     Tag    Clase, Primitivo/Constructor, Tipo
                  (Hex)
---------------- ----- ----------------------------------------
Codificación en 1 Octeto:
00    0   00001:  01    Universal, Primitivo,  BOOLEAN
00    0   00010:  02    Universal, Primitivo,  INTEGER
00    0   00011:  03    Universal, Primitivo,  BIT STRING
00    0   00100:  04    Universal, Primitivo,  OCTET STRING
00    0   00101:  05    Universal, Primitivo,  NULL
00    0   00110:  06    Universal, Primitivo,  OBJECT IDENTIFIER
00    0   00111:  07    Universal, Primitivo,  ObjectDescriptor
...
00    1   11110:  30    Universal, Constructor, SEQUENCE,SEQUENCE OF


Codificación en 2 Octetos:
Octeto 1: 00 1 11111,
Octeto 2:
0     0011111:    31    Universal, Constructor, SET, SET OF
0     0100000:    32    Universal, Constructor, NumericString
0     0100001:    33    Universal, Constructor, PrintableString
...
0     0100100:    36    Universal, Constructor, IA5String
0     0100101:    37    Universal, Constructor, UTCTime
0     0100110:    38    Universal, Constructor, GeneralizedTime
```

- Longitud (L).
  Longitud del valor, en octetos. Existen tres formas de codificar la longitud del valor:

  - Forma Corta.
    Se usa un solo octeto con el bit más significativo en 0. Con esta codificación, la longitud en octetos del valor, puede tomar de 0 a 127 octetos.
  - Forma Larga.
    El bit más significativo del primer octeto tiene un 1. Los siete bits restantes del primer octeto almacenan la longitud de la longitud, es decir el número de octetos que almacenan la longitud (del 1 al 126) del valor en octetos, exceptuando el valor 127 reservado para ampliaciones futuras.



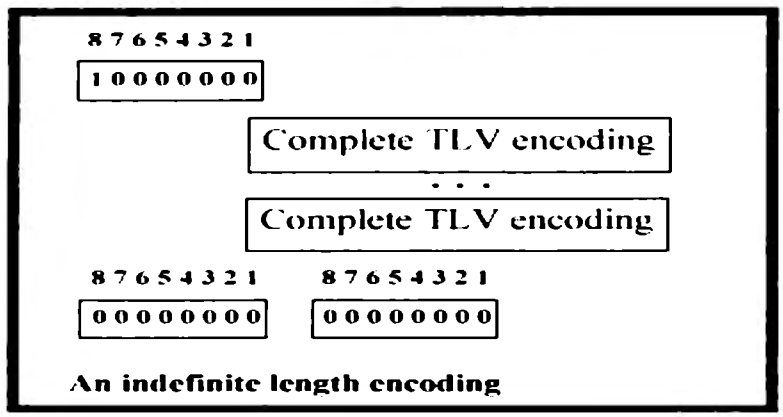
Opciones para codificar una longitud de 5

Con una longitud que puede almacenarse hasta en 126 octetos, la longitud que puede codificarse es inmensa.

- Forma Indefinida.

  La forma indefinida se usa en tipos constructores, es decir que contienen series de valores TLV. En esta codificación el bit más significativo del primer octeto también tiene un 1, pero los bits restantes de este octeto almacenan 0000000.

  Después del primer octeto hay una serie de codificaciones TLV completas, delimitadas al final por un par de octetos en ceros según se muestra en la siguiente figura.



- Valor (V).

  La codificación del valor está íntimamente relacionada con el tipo de dato. En la lista siguiente se muestra la codificación correspondiente a algunos de los tipos de datos, incluyendo su codificación de Tipo, Longitud y Valor.

	Tipo			Longitud	Valor
	Class	P/C	Tag	(0: Corta)	
NULL:	00	0	00101	0 0000000	vacío
TRUE:	00	0	00001	0 0000001	FF Hex
FALSE:	00	0	00001	0 0000001	00 Hex
INTEGER (127):	00	0	00010	0 0000001	7F Hex
INTEGER (128):	00	0	00010	0 0000010	00 80 Hex
INTEGER (-128):	00	0	00010	0 0000001	80 Hex

  En el caso de la codificación del valor entero, éste se realiza en el menor número de octetos, usando el complemento a dos para codificar valores negativos.

Este estudio sobre el lenguaje ASN.1, sienta las bases para analizar las especificaciones relacionadas con el protocolo SNMP.

2.3. SNMP, un Protocolo Simple para la Administración de Redes.

2.3.1. El Modelo Arquitectónico.

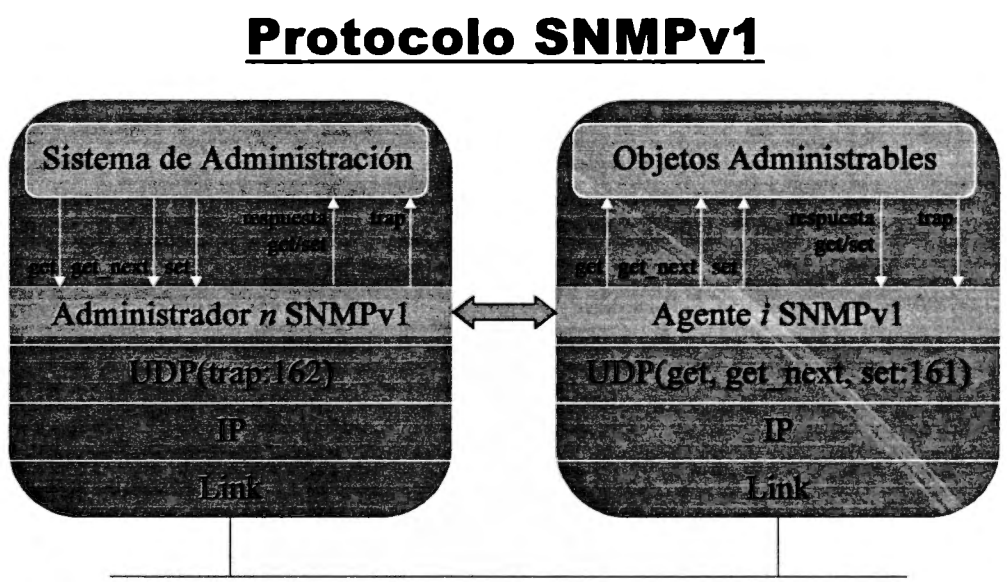
SNMP es un protocolo simple por medio del cual se puede inspeccionar y modificar información administrativa de un elemento administrable de la red.

La especificación de la versión SNMPv1, se encuentra en los siguientes documentos:

- RFC 1155 - Structure of Management Information (SMI) [13].
- RFC 1157 - Simple Network Management Protocol (SNMP) [14] y
- RFC 1213 - Management Information Base II (MIB-2) [15].

La arquitectura de SNMP implica:

- Un Conjunto de Estaciones Administradoras de Red.
  Estas estaciones ejecutan sistemas de administración para supervisar y controlar elementos administrables de red.
- Un Conjunto de Elementos Administrables de Red.
  Estos elementos contienen agentes para responder a los requerimientos de las estaciones administradoras de red.



El protocolo SNMP permite intercambiar mensajes con información administrativa entre las estaciones administradoras de red y los agentes en los elementos de red [16].

Esta arquitectura permite administrar las redes de manera simple, con mínimas restricciones, con facilidades de extensibilidad y con independencia de la plataforma operativa.

2.3.2. Los Elementos de la Arquitectura.

Los elementos de la arquitectura SNMP que articulan una solución al problema de la administración de redes son:

- Alcance de la Información Administrativa.
  La información que manejan las operaciones SNMP son únicamente instancias de los objetos no-agregados (primitivos o escalares) definidos en MIB.

- Representación de la Información Administrativa.
  La información se representa por medio de un subconjunto del lenguaje ASN.1 especificado en la definición de tipos no-agregados en SMI. El hacer uso de un subconjunto de ASN.1, contribuye a la simplicidad de SNMP.

- Operaciones Soportadas Sobre la Información Administrativa.
  Todas las funciones administrativas en el agente se realizan con dos tipos de operaciones: inspecciones (operación get) y modificaciones (operación set) de variables (instancias). Esto evita la inclusión de comandos imperativos específicos con una semántica particular. La estrategia para supervisar el estado de la red se realiza por medio de consultas (polling) usando mensajes de solicitud-respuesta y por medio de mensajes no solicitados (traps) que notifican sobre ciertos eventos del agente. Las notificaciones podrían usarse para guiar la frecuencia y el enfocamiento de las consultas.

- Forma de Intercomunicación en el Protocolo.
  La comunicación entre las entidades SNMP se realiza a través del intercambio de mensajes. Para minimizar la complejidad del agente, el transporte requerido es un servicio de datagramas no confiable (UDP) y cada mensaje se representa completa e independientemente en un solo datagrama.

- Definición de las Relaciones Administrativas.
  - Entidades SNMP.
    Entidades residentes en las estaciones administrativas y elementos administrables de red que se comunican usando SNMP.
  - Comunidad SNMP.
    Un agente SNMP con algún conjunto de entidades SNMP. Cada comunidad se identifica con una cadena de octetos.
  - Mensaje SNMP Auténtico.
    Mensaje originado por una entidad SNMP que pertenece a la comunidad especificada en el mensaje. El conjunto de reglas para identificar un mensaje como auténtico se denomina esquema de autentificación y su

implementación se denomina servicio de autentificación. En SNMPv1 el esquema de autentificación es trivial.

- Vista MIB.
  Subconjunto de objetos de MIB que pertenecen a un determinado elemento administrable de la red. Cada tipo de objeto en MIB tiene definido su modo de acceso.
- Modo de Acceso de la Comunidad.
  Modo de acceso que se asigna al perfil de la comunidad, pudiendo tener los valores READ-ONLY o READ-WRITE.
- Perfil de la Comunidad.
  Vista MIB con el Modo de Acceso de la Comunidad. El acceso a cada variable de MIB sigue las siguientes reglas:
  - Variable en MIB con Acceso None.
    Variable no disponible.
  - Variable en MIB con Acceso Read-Write/Write-Only y Acceso de la Comunidad READ-WRITE.
    Variable disponible para operaciones get, set y trap.
  - Otras combinaciones.
    Variable disponible para operaciones get y trap.
- Política de Acceso.
  Comunidad SNMP con su Perfil. Las relaciones administrativas entre las entidades SNMP se definen en términos de Políticas de Acceso.
- Política de Acceso Proxy.
  Política en la cual la vista MIB no pertenece al elemento de red en el cual el agente reside para una determinada comunidad SNMP. El agente asociado se denomina agente proxy (apoderado).

- Identificación de las Instancias.
  - Nombre Únicos de Variables.
    Las referencias a los objetos MIB se realizan por nombres de variables únicos, por lo que cualquier referencia a cualquier tipo de objeto definido en MIB solo se resolverá a una instancia de tal tipo.
  - Identificación Diferente de Versiones.
    Si se manejan varias versiones MIB, cada versión se identifica de manera diferente (MIB, MIB-2, etc.) por lo que no existe ambigüedad en la identificación de las instancias.
  - Nombres de Variables como OBJECT IDENTIFIERs.
    Las instancias de cualquier tipo de objeto en MIB se identifican en las operaciones SNMP con el nombre de una variable. El nombre de una variable es un OBJECT IDENTIFIER de la forma *t.i* donde *t* es el nombre de un tipo de objeto no agregado definido en MIB e *i* es un fragmento de un OBJECT IDENTIFIER que identifica a la instancia de manera específica.
  - Identificación de Instancias.
    La identificación de instancias representativas se muestra a continuación.

- Objetos con instancia única, t.0. Los objetos con una única posible instancia se identifican agregando un 0. Por ejemplo, el tipo de objeto no agregado sysDescr se identifica como:

```
iso org dod internet mgmt mib system sysDescr
 1   3   6     1      2    1    1       1
```

  Como sysDescr solo tiene una posible instancia (la descripción del sistema), la identificación específica de la instancia es el subidentificador 0, por lo que su instancia se identifica como: 1.3.6.1.2.1.1.1.0
- Objetos de ifTable. Para cada tipo de objeto (ifIndex, ifDescr, ifType, etc.) con el prefijo .ifTable.ifEntry, asociado a la interfaz n (1, 2, etc.), su instancia se identifica como:
  `mgmt.mib-2.interfaces.ifTable.ifEntry.tipo.n`
- Objetos de atTable. Para cada tipo de objeto (atIfIndex, atPhysAddress, atNetAddress) con el prefijo .atTable.atEntry, asociado a la interfaz n (1, 2, etc.) y a la dirección IP A.B.C.D, su instancia se identifica como:
  `mgmt.mib-2.at.atTable.atEntry.tipo.n.1.A.B.C.D`
- Objetos de ipAddrTable. Para cada tipo de objeto (ipAdEntAddr, ipAdEntIfIndex, ipAdEntNetMask, ipAdEntBcastAddr) con el prefijo .ipAddrTable.ipAddrEntry, asociado con la dirección IP A.B.C.D, su instancia se identifica como:
  `mgmt.mib-2.ip.ipAddrTable.ipAddrEntry.tipo.A.B.C.D`
- Objetos de ipRoutingTable. Para cada tipo de objeto (ipRouteDest, ipRouteIfIndex, ipRouteMetric1, ipRouteNextHop, etc.) con el prefijo .ipRoutingTable.ipRouteEntry, asociado con la dirección IP A.B.C.D, su instancia se identifica como:
  `mgmt.mib-2.ip.ipRoutingTable.ipRouteEntry.tipo.A.B.C.D`
- Objetos de tcpConnTable. Para cada tipo de objeto (tcpConnState, tcpConnLocalAddress, tcpConnLocalPort, tcpConnRemAddress, tcpConnRemPort) con el prefijo .tcpConnTable.tcpConnEntry, asociado con la dirección local IP A.B.C.D, el puerto local pl, la dirección remota IP W.X.Y.Z y el puerto remoto pr, su instancia se identifica como:

```
mgmt.mib-2.tcp.tcpConnTable.tcpConnEntry.
                    tipo.A.B.C.D.pl.W.X.Y.Z.pr
```

  Es de observar que los objetos tcpConnTable son dinámicos, solo existen durante una conexión TCP.

2.3.3. El Protocolo SNMP.

SNMP es un protocolo de aplicación por el cual las variables MIB de un agente se pueden inspeccionar o modificar. La comunicación entre entidades pares (peers) del protocolo se realiza intercambiando mensajes, los cuales se representan de manera completa e independiente dentro de un solo datagrama UDP usando las Reglas de Codificación BER de ASN.1. Una entidad del protocolo que actúe como agente, recibirá los mensajes SNMP en el

puerto UDP 161. Una entidad del protocolo que actúe como administrador, recibirá las notificaciones en el puerto UDP 162.

Un mensaje SNMP consta de [17]:

- Identificador de Versión,
- Nombre de una Comunidad SNMP y
- Unidad de Datos del Protocolo (PDU, Protocol Data Unit).
  - GetRequest-PDU
  - GetNextRequest-PDU
  - GetResponse-PDU
  - SetRequest-PDU
  - Trap-PDU

La parte del protocolo SNMPv1 que especifica el Mensaje y las Unidades de Datos del Protocolo (PDUs), es la siguiente:

```
RFC1157-SNMP DEFINITIONS ::=
   BEGIN
      IMPORTS ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks
              FROM RFC1155-SMI;
      -- top-level message
      Message ::= SEQUENCE {version   INTEGER {version-1(0)},
                            community OCTET STRING,
                            data      ANY
                           }
      -- protocol data units
      PDUs ::= CHOICE {get-request        GetRequest-PDU,
                       get-next-request   GetNextRequest-PDU,
                       get-response       GetResponse-PDU,
                       set-request        SetRequest-PDU,
                       trap               Trap-PDU
                      }
```

Cabe mencionar que en el RFC 1157 Sección 4.1.1. existe un error, ya que se hace referencia a 6 PDUs cuando solo están definidos 5.

2.3.4. La Transmisión y la Recepción de Mensajes.

Las acciones necesarias de una entidad del protocolo para transmitir un mensaje, son:

1. Construir el PDU.
   Construir el PDU apropiado como un objeto ASN.1.
2. Autentificar el PDU.
   Enviar al servicio de autentificación el objeto PDU anterior, el nombre de la comunidad y la dirección de transporte (formada por la dirección IP y el puerto UDP) tanto de origen como de destino. El servicio de autentificación regresa otro objeto ASN.1.
3. Construir el Mensaje SNMP.
   Construir un mensaje ASN.1 usando la versión de SNMP, el nombre de la comunidad y el objeto ASN.1 anterior.

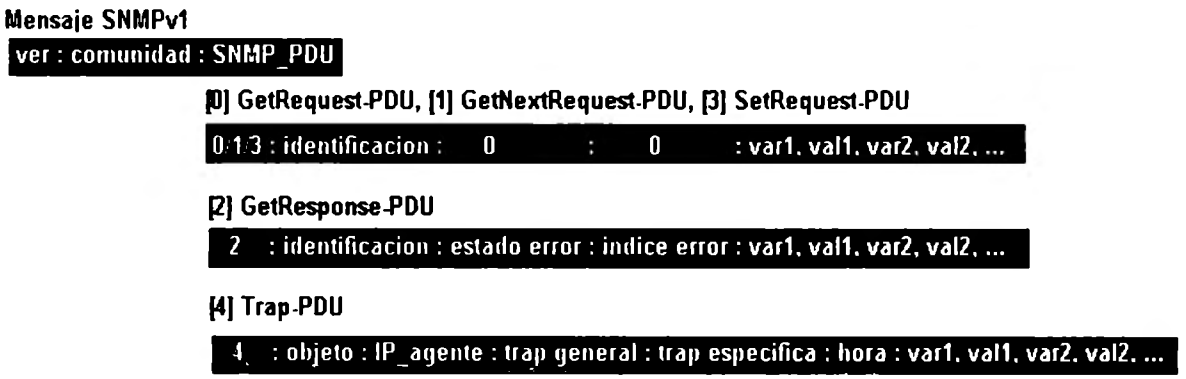
4. Instanciar y Transmitir el Mensaje.
   Instanciar (serializar) este último objeto ASN.1 usando las Reglas de Codificación de ASN.1 y enviarlo usando el servicio de transporte UDP a la entidad par del protocolo.

Las acciones de una entidad del protocolo que recibe un mensaje, son:

1. Construir un Objeto ASN.1 correspondiente al Mensaje.
   Analizar el datagrama entrante para construir un objeto ASN.1 correspondiente a un mensaje. Si el análisis falla, el datagrama se descarta y no se realiza ninguna otra acción.
2. Verificar Versión.
   Verificar si la entidad receptora soporta el número de versión. Si no, el datagrama se descarta y no se realiza ninguna otra acción.
3. Construir un Objeto ASN.1 Autentificado.
   Autentificar el mensaje enviando al servicio de autentificación el nombre de la comunidad, los datos del mensaje y las direcciones de transporte del datagrama. El servicio de autentificación, regresará otro objeto ASN.1 o una indicación de falla de autentificación. La entidad puede generar una notificación (trap) en caso de falla. Si existe una falla, el datagrama se descarta y no se realiza ninguna otra acción.
4. Construir un PDU, Procesarlo y Emitir Respuesta.
   Analizar al objeto ASN.1 regresado por el servicio de autentificación para construir un objeto ASN.1 correspondiente a un PDU. Si el análisis falla, el datagrama se descarta y no se realiza ninguna otra acción, pero si el análisis tiene éxito, se selecciona el perfil correspondiente a la comunidad SNMP y se procesa el PDU. Si se genera un mensaje resultado del procesamiento del PDU, se envía a la dirección de origen del mensaje.

2.3.5. PDUs, las Unidades de Datos del Protocolo.

Un mensaje SNMP consta de los campos de versión, comunidad y PDU.



Para especificar a las primeras cuatro unidades de datos PDU, las que operan bajo el modelo solicitud/respuesta, el protocolo define lo siguiente:

```
-- PDUs
GetRequest-PDU     ::=  [0]  IMPLICIT PDU
GetNextRequest-PDU ::=  [1]  IMPLICIT PDU
GetResponse-PDU    ::=  [2]  IMPLICIT PDU
SetRequest-PDU     ::=  [3]  IMPLICIT PDU

PDU ::= SEQUENCE {request-id        INTEGER,
                  error-status      INTEGER {noError(0),
                                             tooBig(1),
                                             noSuchName(2),
                                             badValue(3),
                                             readOnly(4),
                                             genErr(5)
                                            },
                  error-index       INTEGER,
                  variable-bindings VarBindList
                 }


-- variable bindings
VarBind ::= SEQUENCE {name  ObjectName,
                      value ObjectSyntax
                     }
VarBindList ::=  SEQUENCE OF VarBind

END
```

RequestID identifica al mensaje y permite que una entidad SNMP pueda correlacionar las respuestas recibidas con sus requerimientos solicitados.

ErrorStatus indica si el mensaje fue procesado con ó sin excepciones. Si se instancia con un valor diferente a 0, significa que sucedió una excepción y su valor indica el tipo de excepción. ErrorIndex indica a la variable, dentro de la lista de variables, que fué la causante de la excepción.

Se usa el término variable para referirse a una instancia de un objeto administrable. VarBind (variable binding) define la secuencia de pares formada por el nombre de una variable y su valor. VarBindList es una lista de VarBinds.

Si una variable no requiere su valor (como en GetRequest-PDU), se usa el valor ASN.1 NULL para indicar el valor de la variable en la lista VarBindList.

Consideremos a cada una de las PDU:

- GetRequest-PDU.
  Se genera por la entidad administradora que requiere información de un agente. La entidad receptora o agente responde a tal requerimiento de acuerdo a las siguientes reglas:

  1. Si un objeto nombrado en VarBindList no existe, el receptor enviará al transmisor original, un mensaje GetResponse-PDU similar al PDU recibido, con la diferencia

de que el campo de error-status se instanciará como noSuchName y el valor del campo error-index señalará al objeto no existente.

2. Si un objeto nombrado en VarBindList es un tipo agregado, según se define en SMI, el receptor enviará una respuesta igual a la anterior.
3. Si el tamaño del mensaje GetREsponse-PDU generado, excede alguna limitación local, el receptor enviará al transmisor original un mensaje GetResponse-PDU idéntico al PDU recibido, con la diferencia de que el campo de error-status se instanciará como tooBig y el valor del campo error-index será 0.
4. Si por alguna razón diferente a las anteriores, el valor de un objeto no puede recuperarse en la entidad receptora, el receptor enviará al transmisor original un mensaje GetResponse-PDU idéntico al PDU recibido, con la diferencia de que el campo de error-status se instanciará como genErr y el valor del campo error-index señalará al objeto correspondiente.
5. Si ninguna de las reglas anteriores aplica, la entidad receptora enviará al transmisor original un mensaje cuya parte GetResponse-PDU constará de:
   - request-id: Igual al del PDU recibido.
   - error-status: Se instanciará con noError.
   - error-index: Su valor será 0.
   - variable-bindings: Por cada variable en VarBindList, se responderá con tal variable y su valor.

- GetNextRequest-PDU.

  Se genera por la entidad administradora que requiere información de un agente y es similar a GetRequestPDU excepto por la indicación del tipo de PDU. La entidad receptora o agente responde a tal requerimiento de acuerdo a las siguientes reglas:

  1. Si un objeto nombrado en VarBindList no precede lexicográficamente al nombre de otro objeto disponible para operaciones get en la Vista MIB, el receptor enviará al transmisor original, un mensaje GetResponse-PDU similar al PDU recibido, con la diferencia de que el campo de error-status se instanciará como noSuchName y el valor del campo error-index señalará al objeto nombrado.
  2. Si el tamaño del mensaje GetREsponse-PDU generado, excede alguna limitación local, el receptor enviará al transmisor original un mensaje GetResponse-PDU similar al PDU recibido, con la diferencia de que el campo de error-status se instanciará como tooBig y el valor del campo error-index será 0.
  3. Si por alguna razón diferente a las anteriores, el valor del objeto lexicográficamente sucesor no puede recuperarse en la entidad receptora, el receptor enviará al transmisor original un mensaje GetResponse-PDU idéntico al PDU recibido, con la diferencia de que el campo de error-status se instanciará como genErr y el valor del campo error-index señalará al objeto correspondiente.
  4. Si ninguna de las reglas anteriores aplica, la entidad receptora enviará al transmisor original un mensaje cuya parte GetResponse-PDU constará de:
     - request-id: Igual al del PDU recibido.
     - error-status: Se instanciará con noError.
     - error-index: Su valor será 0.

- variable-bindings: Por cada variable en VarBindList, se responderá con una variable y su valor que en el orden lexicográfico sea el inmediato sucesor de los nombres disponibles para operaciones get en la Vista MIB.

- GetRespond-PDU.
  Se genera por el agente que recibe de la entidad administradora un mensaje que contiene GetRequest-PDU, GetNextRequest-PDU o SetRequest-PDU y es similar a GetRequestPDU excepto por la indicación del tipo de PDU. La entidad receptora o agente genera GetRespond-PDU de acuerdo a las reglas indicadas para responder a GetRequest-PDU, GetNextRequest-PDU y SetRequest-PDU.

- SetRequest-PDU.
  Se genera por la entidad administradora para solicitar a un agente que se modifique el valor de cada una de las variables listadas y es similar a GetRequestPDU excepto por la indicación del tipo de PDU. La entidad receptora o agente responde a tal requerimiento de acuerdo a las siguientes reglas:

  1. Si un objeto nombrado en VarBindList no está disponible para operaciones set en la Vista MIB, el receptor enviará al transmisor original, un mensaje GetResponse-PDU similar al PDU recibido, con la diferencia de que el campo de error-status se instanciará como noSuchName y el valor del campo error-index señalará al objeto nombrado. Si un objeto nombrado en VarBindList tiene un valor que no es consistente (en tipo, longitud o valor) al requerido para la variable de acuerdo al lenguaje ASN.1, el receptor enviará al transmisor original, un mensaje GetResponse-PDU similar al PDU recibido, con la diferencia de que el campo de error-status se instanciará como badValue y el valor del campo error-index señalará al objeto nombrado.
  2. Si el tamaño del mensaje GetResponse-PDU generado, excede alguna limitación local, el receptor enviará al transmisor original un mensaje GetResponse-PDU idéntico al PDU recibido, con la diferencia de que el campo de error-status se instanciará como tooBig y el valor del campo error-index será 0.
  3. Si por alguna razón diferente a las anteriores, el valor del objeto no puede modificarse en la entidad receptora, el receptor enviará al transmisor original un mensaje GetResponse-PDU idéntico al PDU recibido, con la diferencia de que el campo de error-status se instanciará como genErr y el valor del campo error-index señalará al objeto correspondiente.
  4. Si ninguna de las reglas anteriores aplica, la entidad receptora modificará las variables de acuerdo al valor recibido de manera que su efecto sea simultáneo y enviará al transmisor original un mensaje cuya parte GetResponse-PDU constará de:
     - request-id: Igual al del PDU recibido.
     - error-status: Se instanciará con noError.
     - error-index: Su valor será 0.
     - variable-bindings: Igual al recibido.

- Trap-PDU.

  Opera bajo el modelo de notificación. Para especificar un mensaje Trap-PDU, el protocolo SNMPv1 define que debe contener la siguiente información:

```
Trap-PDU ::=
  [4] IMPLICIT SEQUENCE {enterprise        OBJECT IDENTIFIER,
                         agent-addr        NetworkAddress,
                         generic-trap      INTEGER {coldStart(0),
                                                    warmStart(1),
                                                    linkDown(2),
                                                    linkUp(3),
                                                    authenticationFailure(4),
                                                    egpNeighborLoss(5),
                                                    enterpriseSpecific(6)
                                                   },
                         specific-trap     INTEGER,
                         time-stamp        TimeTicks,
                         variable-bindings VarBindList
                        }
```

El campo enterprise identifica al objeto que emite la notificación y agent-addr es su dirección IP. generic-trap es un conjunto de notificaciones predefinidas y specific-trap permite definir otro tipo de notificaciones. time-stamp indica el tiempo desde la última reinicialización del agente hasta la generación de la notificación. La lista de pares nombre-valor, VarBindList, depende de la implementación.

La interpretación del campo generic-trap es la siguiente:

- coldStart.
  El emisor se reinicializó y pudo haberse alterado la configuración del agente o la configuración del protocolo.
- warmStart.
  El emisor se reinicializó pero no se alteró la configuración del agente, ni la configuración del protocolo.
- linkDown.
  El emisor notifica una falla de un enlace representado en su configuración.
- linkUp.
  El emisor notifica la activación de un enlace representado en su configuración.
- authenticationFailure.
  El emisor notifica que recibió un mensaje no auténtico. La implementación debe permitir la emisión opcional de esta notificación.
- egpNeighborLoss.
  El emisor notifica que un vecino EGP está inactivo y la comunicación con él se ha perdido.
- enterpriseSpecificTrap.
  El emisor notifica que un evento específico (enterprise-specific) ha ocurrido. El campo specific-trap contiene el valor de la notificación.

Trap-PDU se genera por un agente a solicitud de su aplicación SNMP, cuando sucede un evento predeterminado y se envía como un mensaje de notificación a un

administrador. El administrador o administradores a quienes se envía la notificación, están determinados por la aplicación SNMP.

La unidad administradora que recibe el mensaje que contiene un Trap-PDU, envía su contenido a su aplicación SNMP para su procesamiento.

2.4. SMI, la Norma para Describir a los Objetos.

2.4.1. Los Elementos Descriptores.

SMI (Structure of Management Information) describe el esquema de identificación y las estructuras para definir la información que se usa en la administración de redes basadas en el conjunto de protocolos TCP/IP. Esta norma se define en el RFC 1155. La norma incluye la descripción de un modelo de objetos de información para la administración de redes y la descripción de un conjunto de tipos de datos genéricos para describir la información administrativa. Ambas descripciones usan el Lenguaje ASN.1. La lista de objetos administrables se encuentra en MIB y la especificación del protocolo en SNMP. SMI atiende los problemas organizacionales y la política administrativa.

El RFC 1155 se desarrolló bajo la idea de la simplicidad y la extensibilidad, ambas motivadas por el reconocimiento de los autores a la falta de profundidad y amplitud en el estudio de la administración de redes. La simplicidad redujo al mínimo las restricciones para el desarrollo futuro de enfoques diferentes, mientras que la extensibilidad maximizó el número de enfoques potenciales para su experimentación.

Los objetos administrables se accesan por medio de un almacén virtual de información denominado Base de Información Administrativa (MIB, Management Information Base), donde cada tipo de objeto tiene los siguientes elementos:

- El nombre, para identificar al objeto,
- La sintaxis, para definir la estructura del objeto y
- La codificación, para representar las instancias del objeto.

2.4.2. El Nombre para Identificar.

Un objeto administrable se identifica con un nombre. El nombre de un objeto se representa de manera única con un OBJECT IDENTIFIER (identificador de objetos). El esquema de identificación que se especifica en SMI es jerárquico y utiliza el concepto de OBJECT IDENTIFIER para modelar esta jerarquía. El OBJECT IDENTIFIER es un medio para identificar objetos sin importar la semántica asociada con el objeto. Por ejemplo, se usa para identificar documentos de normas y para identificar objetos administrables de la red entre otros.

Un OBJECT IDENTIFIER es una secuencia de números enteros que cruzan un árbol computacional global. El árbol consiste de una raíz sin etiqueta conectada a un número de nodos a través de ramas.

El árbol tiene definidos los siguientes elementos en el RFC 1155:

```
root                                OBJECT IDENTIFIER
----                                 -----------------
   ccit(0)                          0
   iso(1)                           1
      org(3)                        1.3
         dod(6)                     1.3.6
            internet(1)             1.3.6.1
               directory(1)         1.3.6.1.1
               mgmt(2)              1.3.6.1.2
                   mib(1)           1.3.6.1.2.1
               experimental(3)      1.3.6.1.3
               private(4)           1.3.6.1.4
                  enterprises(1)    1.3.6.1.4.1
   joint-iso-ccitt(2)               2
```

En la siguiente figura se muestra la parte del RFC 1155 donde se especifica la estructura jerárquica de nombres. Se puede observar por ejemplo, que la rama iso(1) contiene a la subrama org(0), esta a su vez a dod(6) y de manera similar a intertnet(1).

```
RFC1155-SMI DEFINITIONS ::= BEGIN

EXPORTS -- EVERYTHING
        internet, directory, mgmt,
        experimental, private, enterprises,
        OBJECT-TYPE, ObjectName, ObjectSyntax, SimpleSyntax,
        ApplicationSyntax, NetworkAddress, IpAddress,
        Counter, Gauge, TimeTicks, Opaque;

 -- the path to the root

 internet      OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
 directory     OBJECT IDENTIFIER ::= { internet 1 }
 mgmt          OBJECT IDENTIFIER ::= { internet 2 }
 experimental  OBJECT IDENTIFIER ::= { internet 3 }
 private       OBJECT IDENTIFIER ::= { internet 4 }
 enterprises   OBJECT IDENTIFIER ::= { private 1 }
```

2.4.3. La Sintaxis para Definir la Estructura.

La sintaxis de un objeto define su estructura. Para definir la estructura de un objeto se hace uso del tipo ObjectSyntax el cual contiene tipos definidos en ASN.1.

La definición de ObjectSyntax es la siguiente:

```
-- syntax of objects in the MIB
ObjectSyntax ::=
    CHOICE {
            Simple SimpleSyntax,
            application-wide ApplicationSyntax
           }
   SimpleSyntax ::=
       CHOICE {number INTEGER,
               String OCTET STRING,
               Object OBJECT IDENTIFIER,
               Empty  NULL
              }
   ApplicationSyntax ::=
       CHOICE {address   NetworkAddress,
               Counter   Counter,
               gauge     Gauge,
               ticks     TimeTicks,
               arbitrary Opaque
       }
```

Los tipos primitivos de ASN.1, conocidos también como no-agregados, permitidos en SMI son: INTEGER, OCTET STRING, OBJECT IDENTIFIER y NULL. Los tipos INTEGER enumerados no tienen permitido usar el valor 0.

Los tipos constructores de ASN.1, conocidos también como tipos agregados, que se permiten en SMI son:

- Listas.
  Con la sintaxis SEQUENCE {<tipo1>, ..., <tipoN>}, donde cada tipo se resuelve a un tipo no-agregado.
- Tablas.
  Con la sintaxis SEQUENCE OF <entrada>, donde entrada se resuelve a un tipo constructor de listas.

Además de los tipos no-agregados y los tipos agregados anteriores -que finalmente se resuelven a tipos no-agregados-, los objetos pueden tener una sintaxis relacionada con la aplicación. Los tipos definidos para la aplicación SMI son:

```
-- application-wide types
   NetworkAddress ::= CHOICE {internet IpAddress}
   IpAddress      ::= [APPLICATION 0] IMPLICIT OCTET STRING (SIZE (4))
   Counter        ::= [APPLICATION 1] IMPLICIT INTEGER (0..4294967295)
   Gauge          ::= [APPLICATION 2] IMPLICIT INTEGER (0..4294967295)
   TimeTicks      ::= [APPLICATION 3] IMPLICIT INTEGER (0..4294967295)
   Opaque         ::= [APPLICATION 4] IMPLICIT OCTET STRING
```

- NetworkAddress.
  Dirección dentro de una familia de protocolos. Actualmente la única familia de protocolos usada es Internet.
- IpAddress.

Tipo que representa una Dirección Internet de 32 bits. Se representa con un OCTET STRING de longitud 4 en un orden de bytes de red.

- Counter.
  Tipo que repesenta un número natural que se incrementa monotónicamente hasta su valor máximo 2^{32}-1, continúa en cero y vuelve a incrementarse.
- Gauge.
  Tipo que representa un número natural que puede incrementarse o decrementarse pero que no puede incrementarse sobre un valor máximo ni decrementarse bajo un valor mínimo. Su rango de valores es de 0 a 2^{32}-1.
- TimeTicks.
  Tipo que representa un número natural el cual cuenta los centésimos de segundo desde una época. La descripción del objeto identifica la época de referencia. Su valor máximo es 2^{32}-1.
- Opaque.
  Este tipo de dato permite identificar una sintaxis arbitraria. Se codifica como una cadena de octetos (OCTET STRING).

2.4.4. La Codificación para Representar la Instancia.

Una vez que se tiene una instancia de un objeto, su valor se puede transmitir aplicando las Reglas de Codificación BER de ASN.1 a la sintaxis del tipo de objeto. Véase la Sección 2.2.10.

2.5. MIB, el Almacén de los Tipos de Datos.

Los objetos administrables se accesan a través de sus tipos de objetos que están definidos en un almacén de información denominado Base de Información Administrativa ó MIB (Management Information Base).

Debe aclarase que la MIB es una colección de tipos de objetos, no de instancias de objetos. Un tipo de objeto es una definición de una clase de objeto, mientras que una instancia del objeto es un tipo de objeto que ha sido enlazado a un valor.

Las instancias de los objetos no están definidas en la MIB, solo están referenciadas por ella. Una instancia de un tipo de objeto definido en MIB, se identifica por medio de una agregación de información númerica subordinada al OBJECT IDENTIFIER del tipo de objeto. El protocolo es el encargado de referenciar a las instancias.

2.5.1. La Especificación Formal de los Tipos de Objetos.

La especificación formal de cada tipo de objeto -no de las instancias-, se realiza por medio de 5 campos: OBJECT, Syntax, Definition, Access y Status.

- OBJECT.
  Se forma de un nombre textual denominado OBJECT DESCRIPTOR con su correspondiente OBJECT IDENTIFIER que identifica al objeto de manera unívoca.
- Syntax.
  Describe la sintaxis abstracta para el tipo de objeto. La sintaxis se debe resolver a una instancia del tipo ObjectSyntax, comentada en la Sección 2.4.3.
- Definition.
  Descripción textual de la semántica del tipo de objeto.
- Access.
  Facilidad de acceso al tipo de objeto calificada como: read-only, read-write, write-only o not-accessible.
- Status.
  Estado operacional del tipo de objeto calificado como: mandatory, optional u obsolete.

Ejemplo:

```
OBJECT:      atIndex { atEntry 1 }
Syntax:      INTEGER
Definition:  The interface number for the physical address.
Access:      read-write.
Status:      mandatory.
```

Para facilitar la definición formal de tipos de objetos y el uso de herramientas automatizadas de software para procesar la definición de MIB, se usó inicialmente la macro denominada OBJECT-TYPE definida en el RFC 1155, para indicar de manera formal, las características de un tipo de objeto (nótese la falta del campo Definition):

```
-- definition of object types
OBJECT-TYPE MACRO ::=
BEGIN
    TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
                      "ACCESS" Access
                      "STATUS" Status
    VALUE NOTATION ::= value (VALUE ObjectName)
    Access ::=   "read-only"
               | "read-write"
               | "write-only"
               | "not-accessible"
    Status ::=   "mandatory"
               | "optional"
               | "obsolete"
END
```

Un ejemplo de aplicación de esta macro, es la siguiente definición:

```
atIndex OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        ::= { atEntry 1 }
```

La definición de los tipos de objetos en la siguiente versión de MIB, MIB-II RFC 1213, usa la macro OBJECT-TYPE definida en el RFC 1212 Concise MIB Definitions [18], la cual amplía la macro OBJECT-TYPE definida previamente en el RFC 1155.

```
RFC1213-MIB DEFINITIONS ::= BEGIN
IMPORTS
   mgmt, NetworkAddress, IpAddress, Counter, Gauge, TimeTicks
      FROM RFC1155-SMI
   OBJECT-TYPE FROM RFC-1212;
```

Un ejemplo de aplicación de la macro OBJECT-TYPE definida en RFC 1212, es el siguiente:

```
sysContact OBJECT-TYPE
           SYNTAX  DisplayString (SIZE (0..255))
           ACCESS  read-write
           STATUS  mandatory
           DESCRIPTION
                   "The textual identification of the contact person
                   for this managed node, together with information
                   on how to contact this person."
           ::= { system 4 }
```

2.5.2. El Agrupamiento de Objetos en las MIBs.

En la Base de Información Administrativa (MIB) se hace referencia a los objetos, los cuales están organizados en grupos. Los grupos son la unidad básica de conformancia: si la semántica de un grupo es aplicable en una implementación, entonces deben implementarse todos los objetos en ese grupo. Los grupos de objetos en MIB-II son:

```
-- groups in MIB-II
system      OBJECT IDENTIFIER ::= { mib-2 1 }
interfaces  OBJECT IDENTIFIER ::= { mib-2 2 }
at          OBJECT IDENTIFIER ::= { mib-2 3 }
ip          OBJECT IDENTIFIER ::= { mib-2 4 }
icmp        OBJECT IDENTIFIER ::= { mib-2 5 }
tcp         OBJECT IDENTIFIER ::= { mib-2 6 }
udp         OBJECT IDENTIFIER ::= { mib-2 7 }
egp         OBJECT IDENTIFIER ::= { mib-2 8 }

-- historical (some say hysterical)
-- cmot     OBJECT IDENTIFIER ::= { mib-2 9 }

transmission OBJECT IDENTIFIER ::= { mib-2 10 }
snmp        OBJECT IDENTIFIER ::= { mib-2 11 }
```

El grupo at se ha calificado como "deprecated", lo cual significa que en esta versión será soportado pero que en la siguiente versión (MIB-III) será probablemente eliminado.

Enseguida se muestran algunos tipos de datos definidos en MIB-II dentro del grupo system. Nótese que ahora se incluye el campo DESCRIPTION en la notación formal (consecuencia de la macro OBJECT-TYPE definida en RFC 1212):

```
sysContact OBJECT-TYPE
           SYNTAX  DisplayString (SIZE (0..255))
           ACCESS  read-write
           STATUS  mandatory
           DESCRIPTION
                   "The textual identification of the contact person
                   for this managed node, together with information
                   on how to contact this person."
           ::= { system 4 }

   sysName OBJECT-TYPE
           SYNTAX  DisplayString (SIZE (0..255))
           ACCESS  read-write
           STATUS  mandatory
           DESCRIPTION
                   "An administratively-assigned name for this
                   managed node.  By convention, this is the node's
                   fully-qualified domain name."
           ::= { system 5 }

sysLocation OBJECT-TYPE
           SYNTAX  DisplayString (SIZE (0..255))
           ACCESS  read-write
           STATUS  mandatory
           DESCRIPTION
                   "The physical location of this node (e.g.,
                   `telephone closet, 3rd floor')."
           ::= { system 6 }
```

Para conocer a cada tipo de dato definido en MIB-II, consúltese el RFC 1213.

2.5.3. La Actualización de las MIBs.

El avance en el uso del Protocolo SNMP requiere de la actualización de las MIBs. Los tipos de objetos de las MIBs se incluyen bajo los siguientes criterios:

- Incluir objetos esenciales para la administración de fallas o de la configuración.
- Incluir objetos cuyo control no cause daños graves en caso de una operación errónea.
- Incluir objetos con evidencia de uso y utilidad.
- Limitar el número máximo de tipos de objetos a 100.
- Evitar variables redundantes.
- Evitar objetos específicos a alguna implementación.

Para incluir nuevos objetos en las MIBs se tienen 3 mecanismos de extensibilidad:

- Adición de objetos populares pero no normalizados a través de la rama experimental,
- Adición de objetos privados a través de la rama enterprises y
- Adición de nuevos objetos en versiones nuevas de MIB.

Una nueva versión de MIB vuelve obsoletas a las anteriores pero mantiene la compatibilidad con ellas. Para mantener la compatibilidad, el prefijo de OBJECT IDENTIFIER en MIB-II es similar al prefijo en MIB-I:

```
--  MIB-II (same prefix as MIB-I)
mib-2       OBJECT IDENTIFIER ::= { mgmt 1 }
```

Para manejar los tipos de objetos anteriores en una MIB nueva, se tienen 3 posibilidades:

- Declararlos Obsoletos.
  Declarar a los tipos de objetos anteriores, si es necesario, como obsoletos, pero no borrar sus nombres.
- Extender su Definición.
  Aumentar la definición de un tipo de objeto correspondiente a una lista incluyendo tipos de objeto no-agregados.
- Definir Nuevos Tipos de Objetos.
  Definir tipos de objetos completamente nuevos,

pero no en ningún caso se debe cambiar la semántica de un tipo de objeto previamente definido.

Esto permite implementar de manera sencilla el soporte a diferentes versiones de MIBs. La versión actual de MIB, conocida como MIB-II ó MIB-2 (RFC 1213), soporta a MIB-I (RFC 1156).

El estudio del protocolo SNMP permite introducirse al funcionamiento de la infraestructura para la administración de redes TCP/IP, a fin de contar con el marco teórico necesario para desarrollar la presente investigación.

3. Tcl, un Lenguaje Integrador de Sistemas.

3.1. El Lenguaje Tcl.

La versión 1997 del prestigiado premio anual ACM Software System Award (http://www.acm.org/awards/ssaward.html), se asignó al Lenguaje Tcl/Tk, argumentándose lo siguiente:

> *"For the Tcl scripting language which allows developers to create complex systems from pre-existing components.*
>
> *The embedded Tk provides a simple mechanism for creating graphical user interfaces.*
>
> *Together they make a powerful addition to the software."*

Ganadores anteriores de este premio incluyen a sistemas tan renombrados como TCP/IP, Visicalc, World Wide Web, Unix, PostScript, y SmallTalk. ¿Que características tiene Tcl para ser considerado como un sistema de software de tal influencia? [19].

El desarrollo de los lenguajes de programación se enfocó inicialmente a la arquitectura de la computadora, desarrollándose programas en lenguaje de máquina y en lenguaje ensamblador. Posteriormente la programación contó con lenguajes orientados a la solución de los problemas, conocidos como lenguajes de alto nivel. Los lenguajes de alto nivel se enfocan a cómputos intensivos y soportan estructuras de datos y algoritmos complejos. Son lenguajes tipificados donde es necesario definir cada tipo de estructura de datos utilizada.

Programar en un nivel más alto ha significado en la práctica, disminuir el esfuerzo de programación, logrando tiempos de desarrollo mas cortos y códigos fuente de menor tamaño. Estos lenguajes de alto nivel partieron de la idea de desarrollar código sin reusar el existente. En la actualidad un nuevo paradigma de programación, la programación orientada a objetos, intenta incrementar aún más la productividad con técnicas de reuso de software.

A diferencia de los lenguajes de programación tradicionales, los lenguajes script se enfocan a usar las aplicaciones existentes -programas ejecutables desarrollados generalmente en lenguajes tradicionales- e integrarlas, enfocándose a ordenar la ejecución de las aplicaciones y capturar sus resultados. Para simplificar el intercambio de información entre aplicaciones, los lenguajes script tienden a no tener tipos de datos o a manejar uno solo, las cadenas de caracteres. De hecho, en el interior de la computadora los datos no están tipificados, su interpretación depende del software.

Dado el enfoque de los lenguajes script de servir como "pegamento" de aplicaciones [20], sus capacidades de comunicación son simples y poderosas. Se ha reportado que al usar estos lenguajes se logran tiempos de desarrollo y longitud de código del orden de 1/5 a 1/10 de lo que se logra usando lenguajes tradicionales. Ejemplos de lenguajes script incluyen Perl, Phyton, Rexx, Visual Basic, el Lenguaje Batch de MsDos y los Shells de Unix. La figura siguiente muestra una tabla comparativa entre los lenguajes script más populares en el 2001.

Features		Tcl	Perl	Python	Java Script	Visual Basic
Speed of use	Rapid development	✓	✓	✓	✓	✓
	Flexible, rapid evolution	✓	✓	✓	✓	✓
	Great regular expressions	✓	✓	✓		
Breadth of functionality	Easily extensible	✓		✓		✓
	Embeddable	✓		✓		
	Easy GUIs	✓		✓		✓
	Internet and Web-enabled	✓	✓	✓	✓	✓
Enterprise usage	Cross platform	✓	✓	✓	✓	
	Internationalization support	✓		✓	✓	✓
	Thread safe	✓		✓		✓
	Database access	✓	✓	✓	✓	✓

http://www.scriptics.com/advocacy/

Programar en lenguajes script se considera programar en un *lenguaje de muy alto nivel*, lo que les ha ganado el nombre de *lenguajes de integración de sistemas* [21].

Tcl (Tool Command Language) es un lenguaje script desarrollado por John Ousterhout para controlar y extender aplicaciones. Tcl es básicamente un lenguaje textual y una biblioteca C.

Como lenguaje textual, Tcl está enfocado al envío de comandos a programas interactivos. Tiene facilidades de programación como variables, control de flujo y procedimientos. Los comandos internos de Tcl, conocidos como el núcleo Tcl, pueden ser ampliados con procedimientos de comandos escritos por el usuario.

Como biblioteca (biblioteca C), Tcl se puede incrustar en los programas de aplicación. Esta biblioteca consta de un analizador (parser) del Lenguaje Tcl, rutinas que implementan los comandos internos de Tcl y procedimientos que permiten a la aplicación extender Tcl con nuevos comandos específicos. Cuando Tcl se incrusta como una biblioteca en una aplicación, ésta genera comandos que pasa al analizador Tcl para su ejecución. El analizador obtiene los campos que integran al comando y lo ejecuta directamente si se trata de un comando interno. Si el comando es específico de la aplicación, Tcl llama a la aplicación para que ejecute tal comando, pudiéndose inclusive llegar a invocar recursivamente a Tcl.

Cuando un desarrollador de aplicaciones usa Tcl, puede obtener los siguientes beneficios:

1. Desarrollo rápido de aplicaciones incluyendo aplicaciones gráficas.
   Tcl y sus extensiones permiten un desarrollo rápido de aplicaciones, particularmente aplicaciones gráficas. Tk (Tool Kit) es una extensión del

Lenguaje Tcl que permite usar interfaces gráficas. Cuando se usa el shell *wish (windows shell)*, se pueden ejecutar scripts de Tk además de scripts de Tcl. Con wish se puede programar en un nivel más alto que C o C++, quedando ocultos los detalles de programación gráfica. Además, al ser Tcl/Tk un lenguaje interpretado, la prueba y depuración del código es rápida..

2. Facilidad de aprendizaje.
   La sintaxis estándar de Tcl permite que una vez comprendida la estructura de un comando, sea posible emitir prácticamente cualquier comando ya que su estructura es similar.
3. Facilidad para implementar aplicaciones.
   Para implementar una nueva aplicación, generalmente es suficiente implementar algunos comandos específicos de la aplicación y utilizar las capacidades internas de programación de Tcl, para producir procedimientos complejos. Esto permite obtener un lenguaje script completo que incluye a los comandos internos de Tcl -conocidos como el núcleo Tcl- y a los comandos específicos de la aplicación.
4. Facilidad de comunicación entre aplicaciones.
   Tcl se puede usar como un lenguaje común para la intercomunicación entre aplicaciones tanto locales como remotas bajo el conjunto de protocolos TCP/IP. Los scripts de Tcl se pueden usar como un mecanismo de comunicación que permiten a diferentes aplicaciones trabajar de manera integrada.

Tcl es un lenguaje interpretado por lo que la ejecución de sus aplicaciones no alcanza la velocidad de ejecución de las aplicaciones desarrolladas en un lenguaje compilado. Tcl no es recomendable donde la velocidad afecta sensiblemente al desempeño de la aplicación, como aquellas que realizan cómputos intensivos, pero ante aplicaciones de tamaño mediano o pequeño, ejecutadas en computadoras de alta velocidad y donde la aplicación no es particularmente sensible a la velocidad de ejecución, Tcl tiene un campo especial de desarrollo, con resultados sorprendentes especialmente en el área de administración de redes [22].

El elemento central en Tcl es el intérprete. Un intérprete Tcl consta de un conjunto de comandos, un conjunto de variables e información de estado. Los comandos se interpretan en el contexto de un intérprete específico.

Tcl soporta como único tipo de datos a las cadenas de caracteres (strings). Los comandos, sus argumentos, sus resultados y los valores de sus variables, son cadenas de caracteres. Cuando un comando realiza una operación sobre un argumento numérico o emite un resultado numérico, la información también se transmite como cadenas de caracteres.

Se puede entender en principio la operación de un intérprete Tcl, considerando que realiza operaciones únicamente sobre cadenas de caracteres. Las construcciones Tcl no tienen estructura, solo son cadenas de caracteres y esto hace que su comportamiento sea diferente a las construcciones de los lenguajes estructurados. Las cadenas de caracteres tienen 3 formas comunes de interpretarse: como comandos, como expresiones y como listas.

3.2. Los Comandos.

Una forma de interpretar a las cadenas de caracteres es como una cadena de comandos. Una cadena de comandos Tcl consiste de uno o más comandos separados por un fin de línea o por el signo ";". Cada comando esta formado por un conjunto de campos o palabras separadas por espacios en blanco (espacios o tabuladores). El primer campo debe ser el nombre del comando y si hay campos adicionales serán los argumentos que se pasarán al comando.

El comando puede ser interno, un comando específico de la aplicación o un procedimiento de comandos definido con el comando *proc*. Los argumentos se pasan como cadenas de texto y los comandos son quienes toman la responsabilidad de interpretarlos [23].

Si en un comando, el primer carácter diferente a un espacio en blanco es "#", los caracteres que continúan hasta el fin de línea se consideran un comentario.

3.2.1. El Agrupamiento de Argumentos.

Los argumentos están delimitados normalmente por espacios en blanco, pero si un argumento incluye espacios en blanco o puntos y comas, puede agruparse por medio de comillas dobles o llaves en sus extremos.

La sustitución de comandos, de variables y de diagonales invertidas (temas considerados posteriormente), se permite en agrupamientos con comillas, pero no se permite en agrupamientos con llaves.

Las llaves pueden manejar anidamientos, las comillas no. Cuando las llaves o las comillas están agrupando argumentos, la llave o comilla final puede localizarse en una alguna línea posterior. Los fines de línea (LF en Unix, CRLF en MsDos) intermedios, se incluirán en el agrupamiento junto con todos los caracteres que se encuentren hasta la comilla o llave final.

3.3. Las Expresiones.

Otra forma de interpretar a las cadenas de caracteres en Tcl es considerarlas como expresiones. Algunos comandos de Tcl como *expr*, *for* e *if* tratan a sus argumentos como expresiones y llaman a los procesadores de expresiones de Tcl (Tcl_ExprLong, Tcl_ExprBoolean y otros) para evaluarlas. El manejo de expresiones en Tcl es similar a C. Los operadores en Tcl son un subconjunto de los de C y tienen el mismo significado y precedencia. Las expresiones Tcl soportan operandos no numéricos y comparaciones de cadenas de caracteres.

Una expresión Tcl consta de operadores, operandos y paréntesis. Se pueden usar espacios en blanco y paréntesis para separar a sus elementos. Los operandos se intentan interpretar inicialmente como valores enteros, ya sea en forma decimal, octal (0) o hexadecimal (0x),

después se intentan interpretar como valores de punto flotante según el formato usado en los compiladores ANSI-C y finalmente el operando se interpreta como una cadena de caracteres.

La siguiente lista muestra las posibilidades para interpretar a los operandos de las expresiones:

1. Como un valor numérico entero
2. Como un valor de punto flotante.
3. Como una cadena.
   3.1. Valor de una variable Tcl: $nombre_de_variable.
   3.2. Cadena encerrada entre comillas dobles.
        El procesador de expresiones realiza las sustituciones correspondientes y usa el resultado como operando.
   3.3. Cadena encerrada entre llaves.
        La cadena se usa como operando sin realizarse sustituciones.
   3.4. Cadena encerrada entre paréntesis cuadrados.
        La cadena se interpreta como un comando encerrado entre paréntesis cuadrados, se ejecuta y el resultado se usa como operando.

Cuando se requieren sustituciones en los casos anteriores, éstas deben realizarse por el procesador de expresiones. Puede suceder sin embargo, que el analizador de comandos haya realizado alguna sustitución antes de llamar al procesador de expresiones y se obtengan resultados no deseados. Para evitar que el analizador de comandos realice sustituciones previas, es recomendable encerrar a las expresiones entre llaves.

Los cálculos con enteros se realizan con el tipo de datos *long* de C y los que involucran operaciones de punto flotante con el tipo *double* de C. La conversión entre enteros, punto flotante y cadenas se realiza de manera automática cuando se requiere. En cálculos aritméticos, se usan enteros hasta que se opera algún valor en punto flotante, después de lo cual las operaciones se realizan en punto flotante.

Las cadenas se pueden usar como operandos en los operadores de comparación (<, >, etc.). El evaluador de expresiones trata de realizar las comparaciones considerando inicialmente a las expresiones como valores enteros, después como valores de punto flotante y finalmente como cadenas.

Si un operando en una comparación es una cadena de caracteres y el otro operando es un valor numérico, el valor numérico se convierte a cadena usando el especificador de formatos "*%d*" para enteros y "*%g*" para flotantes, de la función *sprintf* de C.

Para manejar a las expresiones correctamente, es recomendable encerrar la expresión entre llaves, ya que si la expresión contiene espacios en blanco, el intérprete Tcl los entenderá como delimitadores de argumentos y dividirá a la expresión completa en varios argumentos. Si la expresión no se encierra entre llaves, se realizará la sustitución de variables y comandos inmediatamente.

Si se requiere que una expresión se evalúe posteriormente, con las sustituciones correspondientes en el momento de cada evaluación, como es el caso de la expresión para la condición de terminación del comando *for*, se deben usar las llaves para encerrar la expresión.

Los operadores de Tcl son los siguientes y están agrupados por orden decreciente de precedencia:

- ~ !	Menos unario, NOT a nivel bit, NOT lógico. Se aplican a operandos numéricos. ~ se aplica solo a valores enteros.
* / %	Multiplicación, división, residuo. Se aplican a operandos numéricos. % se aplica solo a valores enteros.
+ -	Suma, resta. Se aplican a operandos numéricos.
<< >>	Corrimiento izquierdo, corrimiento derecho. Se aplican a operandos numéricos enteros.
< > <= >=	Menor que, Mayor que, Menor o Igual que, Mayor o Igual que. Operadores de comparación que producen un 1 si la condición es verdadera, o 0 de otra manera.
== !=	Igual, Diferente. Operadores de comparación.
&	AND a nivel bit. Se aplica solo a operandos numéricos enteros.
^	OR Exclusivo a nivel bit. Se aplica solo a operandos numéricos enteros.
|	OR Inclusivo a nivel bit. Se aplica solo a operandos numéricos enteros.
&&	AND Lógico. Produce un 1 si sus operandos son diferentes de cero, o produce un 0 de otra manera. Se aplica a operandos numéricos.
||	OR Lógico. Produce un 0 si sus operandos son cero, o produce un 1 de otra manera. Se aplica a operandos numéricos.
x ? y : z	Si x Entonces y Sino z. Si el operando numérico "x" es diferente de cero, el resultado es el operando "y", de otra manera el resultado es el operando "z".

3.3.1. Las Expresiones Regulares.

Las cadenas son el tipo básico de datos de Tcl. Esto permite comprender el porqué de la riqueza de mecanismos en Tcl para procesarlas. Una manera de mejorar el procesamiento de cadenas es utilizar expresiones regulares.

Las expresiones regulares permiten describir patrones de caracteres de manera compacta. Al evaluar una expresión regular se define una cadena basada en una igualación -match-, de la expresión regular.

Cuando se usan expresiones regulares en los comandos, estos se expresan en forma compacta y se ejecutan de manera más eficiente. Tcl usa a los comandos *regexp* y *regsub* para realizar igualación de cadenas de caracteres.

Una expresión regular está integrada por cero o más *ramas* separadas por el signo "|". Estas ramas se prueban en paralelo, igualando la expresión regular a cualquier igualación de alguna rama:

expresión regular: *rama1* | *rama2* | ... | $rama_i$ | ...

Una rama contiene cero o más *piezas* concatenadas. Si una rama iguala a una expresión, significa que existe una igualación con la primer pieza, seguida de una igualación con la segunda pieza y así hasta la última pieza de la rama:

$rama_i$: $pieza_1 pieza_2 .... pieza_i ....$

Una pieza está formada por un *átomo* con la posibilidad de ser seguida por un caracter "+", "*" o "?". Si el átomo está seguido del caracter "+", puede igualar 1 o más secuencias del átomo. Si está seguido del caracter "*", puede igualar 0 o más secuencias. Si está seguido del caracter "?", la pieza puede igualar a la cadena nula ó al átomo.

$pieza_i$:	*átomo*+	(1 ó más secuencias) ó
	*átomo**	(0 ó más secuencias) ó
	átomo?	(0 ó 1 secuencia)

Un átomo puede ser:

(expresión regular)	Expresión regular entre paréntesis a la que iguala.
[rango]	Un rango al que iguala.
.	Un ".", que iguala a cualquier caracter.
^	Un "^" que iguala la cadena nula al principio de la cadena de entrada.
$	Un "$", que iguala la cadena nula al final de la cadena de entrada.
\	Un "\" seguida por un caracter al que iguala.
caracter	Un caracter al que iguala.

Un rango es una secuencia de caracteres entre paréntesis cuadrados. Iguala a cualquier caracter de la secuencia. Si la secuencia empieza con "^", iguala a cualquier caracter que no se encuentre en el resto de la secuencia. Si dos caracteres están separados por el caracter "-", la secuencia considera el rango ASCII entre los caracteres.

Si una expresión regular puede igualar dos partes de una cadena de caracteres, seleccionará a la primera como la parte igualada.

Otros casos más elaborados, donde se da la posibilidad de seleccionar una de varias posibles igualaciones, por ejemplo cuando la longitud difiere, deben consultarse en el manual del lenguaje.

3.3.2. Los Patrones Globales.

Otro mecanismo para generar cadenas de caracteres de manera compacta son los patrones globales, un lenguaje simple de patrones, usado principalmente para igualar nombres de archivos.

Los patrones globales igualan a cadenas de caracteres usando los siguientes mecanismos:

- ? Iguala a cualquier carácter.
- * Iguala a cero o más caracteres.
- [abc] Iguala un conjunto de caracteres.
- [a-z] Iguala un rango de caracteres.
- \c Iguala a un caracter.

3.4. Las Listas.

Las listas son otra forma de interpretar a las cadenas de caracteres en Tcl. Una lista es una cadena de caracteres agrupada, con una estructura de campos separados por delimitadores. Los mecanismos de agrupación son las comillas dobles y las llaves. Los delimitadores de campos incluyen espacios en blanco (espacios y tabuladores) y el caracter de línea nueva. Aplica también la sustitución de secuencias de diagonales invertidas pero no la sustitución de comandos ni la sustitución de variables.

3.5. Las Sustituciones.

Tcl cuenta con mecanismos para sustituir comandos, variables y secuencias de diagonales invertidas.

Sustituir un comando significa que si un comando se encuentra entre paréntesis cuadrados, se ejecuta y se sustituye por su resultado.

Sustituir una variable significa que si un nombre de variable está antecedida por el signo "$", se sustituye por su valor, siempre y cuando no esté encerrada entre llaves. $nombre_de_variable es equivalente a [set nombre_de_variable].

Existen dos casos especiales de sustitución de variables. El primero caso se da cuando después del nombre de la variable antecedido por el signo "$", continúa un paréntesis. En este caso se considera que se está manejando un arreglo y que el índice se encuentra entre paréntesis: $nombre_de_arreglo(indice). Antes de usar el índice, se realiza la sustitución de comandos y la sustitución de variables sobre la información entre paréntesis, si estas operaciones están indicadas.

El segundo caso especial de sustitución de variables se da cuando al signo "$" le sigue una llave. En este caso el nombre de la variable se encuentra entre las llaves y solo puede

referenciar a variables escalares, no a arreglos: ${nombre_variable}. La sustitución de variables no ocurre en argumentos encerrados entre llaves.

La sustitución de secuencias de diagonales invertidas es un mecanismo que permite incluir caracteres especiales sin que se interpreten en la forma usual, tales como caracteres no imprimibles, comillas dobles, llaves y paréntesis cuadrados. Este mecanismo de escape es similar al usado en el Lenguaje C.

Si la diagonal invertida es seguida por un caracter distinto a los indicados en la lista siguiente, la diagonal invertida se transmite como parte del argumento correspondiente sin ningún procesamiento especial. Si un argumento está encerrado entre llaves, las secuencias de diagonal invertida en su interior no se sustituyen, con la excepción de la secuencia \nl.

La sustitución de secuencias de diagonales invertidas permite generar a la mayoría de los argumentos, sin embargo para argumentos particularmente complejos, puede usarse el comando *format*.

Cuando se tiene una secuencia de caracteres reconocida por Tcl que inicia con la diagonal invertida, se sustituye por el caracter correspondiente según la siguiente lista:

\b	Retroceso	0x08
\f	Alimentación de Forma	0x0C
\n	Línea Nueva	0x0A
\r	Retorno de Carro	0x0D
\t	Tabulador	0x09
\v	Tabulador Vertical	0x0B
\{	Llave izquierda	{
\}	Llave derecha	}
\[	Paréntesis Cuadrado Inicial	[
\]	Paréntesis Cuadrado Final	]
\$	Signo de Moneda	$
\sp	Espacio	
\;	Punto y coma	;
\"	Comillas dobles	"
\\	Diagonal invertida	\
\ddd	Caracter cuyo valor octal es ddd. Si ddd es 000, la secuencia se mapea a una cadena vacía.	
\nl	Tiene el efecto de interpretar dos líneas como una. Se aplica aún si la secuencia ocurre entre llaves.	

3.6. Los Procedimientos.

Tcl permite ampliar sus comandos definiendo procedimientos. Un procedimiento es una cadena de caracteres que contiene comandos Tcl. El procedimiento se invoca de manera similar a un comando interno, tiene un nombre y puede recibir argumentos.

Antes de invocar un procedimiento, se le debe definir en Tcl. El comando *proc* permite definir los procedimientos en Tcl bajo el siguiente formato:

```
proc nombre_del_procedimiento {arg1 arg2 ...} {
        comando Tcl
        comando Tcl

}
```

El procedimiento puede definirse interactivamente ó prepararse en un archivo y cargarse posteriormente en la memoria de trabajo usando el comando *source*.

3.7. Las Variables.

Tcl permite la definición de variables y el uso de sus valores principalmente por medio del comando *set* y la sustitución de variables con "$". Las variables no necesitan ser declaradas. El nombre de una variable puede incluir letras, números y el signo de subrayar.

Tcl soporta escalares y arreglos. Una variable escalar tiene un solo valor, mientras que una variable arreglo tiene un número de elementos los cuales se nombran por un índice y cada elemento tiene su valor. Se usan los paréntesis para referirse a los elementos de un arreglo.

A diferencia de la mayoría de los lenguajes de programación que usan valores numéricos enteros como índices, Tcl soporta índices para los arreglos que pueden ser cadenas de caracteres.

Los arreglos multidimensionales se pueden simular usando índices que contengan valores concatenados.

Las variables pueden tener un alcance global o local a un procedimiento dependiendo de si están nombradas fuera de cualquier procedimiento o dentro de alguno. Las variables locales solo existen en la ejecución del procedimiento y se eliminan cuando el procedimiento termina.

Si se requiere que una variable nombrada en un determinado procedimiento tenga un alcance global, se usa el comando *global* al definir a la variable dentro de tal procedimiento.

Para remover a las variables de la memoria, se usa el comando *unset*.

3.8. Los Comandos Internos.

La siguiente lista muestra una descripción resumida de los comandos internos de Tcl. Al solicitar la ejecución de un comando debe considerarse que es sensible a las mayúsculas y a las minúsculas. Para una explicación completa de los comandos consúltese el manual del lenguaje ó esta referencia [24].

after	Retrasa la ejecución de un comando.
append	Agrega argumentos al valor de una variable. No agrega espacios.
array	Consulta el estado de un arreglo y busca en sus elementos.
binary	Convierte entre datos cadena y binarios.
break	Salida prematura de un ciclo.
catch	Captura errores.
cd	Cambia el directorio de trabajo.
clock	Obtiene la hora y formatea cadenas de fecha.
close	Cierra un flujo I/O abierto.
concat	Concatena argumentos con espacios entre ellos. Divide listas.
console	Controla la consola usada para introducir comandos interactivamente.
continue	Continúa con la siguiente iteración del ciclo.
error	Indica un error.
eof	Verifica el fin de un archivo.
eval	Concatena argumentos y los evalúa como un comando.
exec	Ejecuta un programa.
exit	Termina un proceso.
expr	Evalúa una expresión matemática.
fblocked	Consulta un canal I/O para saber si hay datos disponibles.
fconfigure	Configura y consulta las propiedades de un canal I/O.
fcopy	Copia de un canal I/O a otro.
file	Consulta al sistema de archivos.
fileevent	Registra la llamada de regreso de un I/O manejado por eventos.
flush	Libera la información de salida de los buffers internos de un flujo I/O.
for	Construye un ciclo, similar a la instrucción for de C.
foreach	Construye un ciclo sobre una lista, o listas de valores.
format	Formatea una cadena de manera similar a la instrucción sprintf de C.
gets	Lee una línea de entrada de un flujo I/O.
glob	Expande un patrón para igualar nombres de archivos.
global	Declara variables globales.
history	Controla la historia de la línea de comandos.
if	Condiciona la ejecución de comandos. Permite las claúsulas else y elseif.
incr	Incrementa una variable por una cantidad entera.
info	Consulta el estado del Intérprete Tcl.
interp	Crea Intérpretes Tcl adicionales.
join	Concatena elementos de una lista con un separador de cadenas dado.
lappend	Agrega elementos al final de una lista.
lindex	Busca y recupera un elemento de una lista.
linsert	Inserta elementos en una lista.
list	Crea una lista de los argumentos.
llength	Regresa el número de elementos de la lista.
load	Carga bibliotecas compartidas que definen comandos Tcl.
lrange	Regresa un rango de una lista de elementos.
lreplace	Reemplaza elementos de una lista.
lsearch	Busca un elemento de una lista que iguale un patrón.
lsort	Ordena una lista.
namespace	Crea y manipula espacios de nombres.
open	Abre un archivo o un canal de comunicación unidireccional de procesos para I/O.
package	Proporciona o requiere paquetes de código.
pid	Regresa la identificación del proceso.
proc	Define un procedimiento Tcl.
puts	Envía una cadena a un flujo I/O.
pwd	Regresa el directorio de trabajo actual.
read	Lee bloques de caracteres de un flujo I/O.
regexp	Iguala expresiones regulares.
regsub	Realiza sustituciones basadas en expresiones regulares.

rename	Cambia el nombre de un comando Tcl.
return	Regresa un valor desde un procedimiento.
scan	Analiza una cadena de acuerdo a una especificación de formato.
seek	Configura el corrimiento de una búsqueda de un flujo I/O.
set	Asigna y recupera el valor de una variable.
socket	Abre una conexión de red TCP/IP.
source	Evalúa los comandos Tcl almacenados en una archivo.
split	Divide una cadena en una lista de elementos.
string	Realiza operaciones sobre cadenas.
subst	Realiza sustituciones sin evaluación de comandos.
switch	Ramifica en varias opciones.
tell	Regresa el corrimiento actual de búsqueda de un flujo I/O.
time	Mide el tiempo de ejecución de un comando.
trace	Monitorea las asignaciones de las variables.
unknown	Maneja comandos desconocidos.
unset	Borra variables.
uplevel	Ejecuta un comando en un ambiente diferente.
upvar	Hace referencia a una variable en un ambiente diferente.
variable	Declara variables que nombran espacios.
vwait	Espera que una variable sea modificada.
while	Construye un ciclo.

3.9. Las Funciones Matemáticas Internas.

La siguiente lista es un resumen de las funciones matemáticas internas.

abs(x)	Valor absoluto de x.
acos(x)	Arco coseno de x.
asin(x)	Arco seno de x.
atan(x)	Arco tangente de x.
atan2(y,x)	Valor angular th de la conversión de coordenadas rectangulares (x,y) a polares (r,th).
ceil(x)	El más pequeño valor entero mayor o igual a x.
cos(x)	Coseno de x.
cosh(x)	Coseno hiperbólico de x.
double(x)	Promueve x a punto flotante.
exp(x)	Exponencial. Valor de e^x.
floor(x)	El más grande valor entero menor o igual a x.
fmod(x,y)	Residuo en punto flotante de x/y.
hypot(x,y)	Calcula sqrt(x*x + y*y).
int(x)	Trunca x a un valor entero.
log(x)	Logaritmo natural (en base e) de x.
log10(x)	Logaritmo en base 10 de x.
pow(x,y)	Valor de x a la potencia de y, x^y.
rand()	Regresa un valor aleatorio de punto flotante entre 0.0 y 1.0.
round(x)	Redondex a un valor entero.
sin(x)	Seno de x.
sinh(x)	Seno hiperbólico de x.
sqrt(x)	Raíz cuadrada de x.
srand(x)	Define la raíz x del generador de números aleatorios.
tan(x)	Tangente de x.
tanh(x)	Tangente hiperbólica de x.

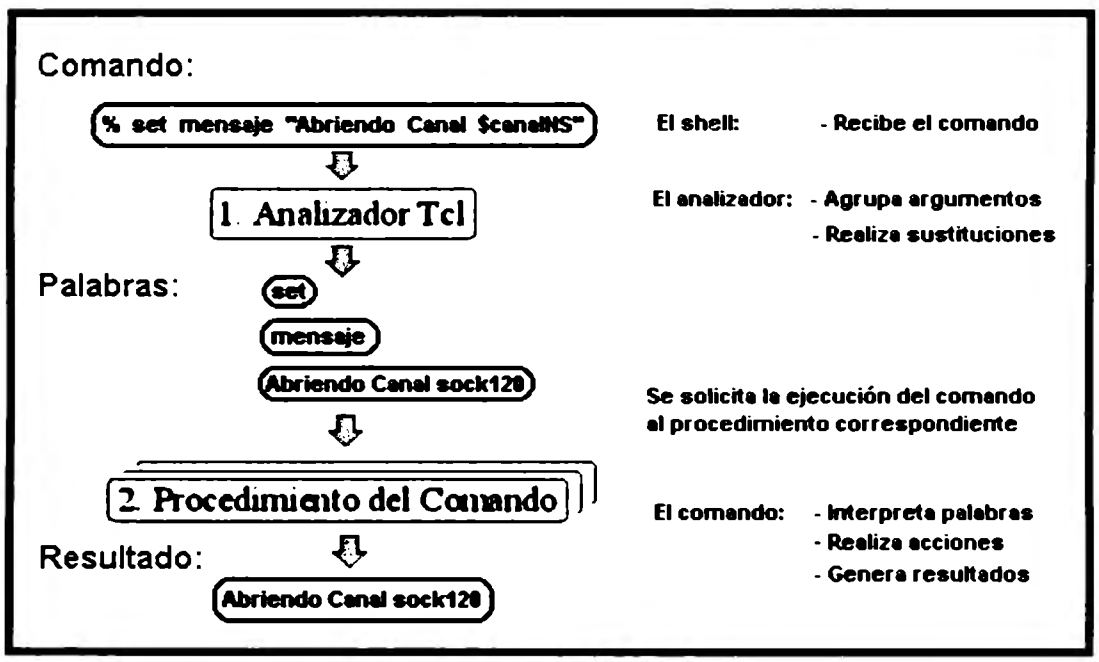
3.10. La Ejecución de Comandos.

Tcl soporta dos shells para intercomunicarse con el usuario: tclsh y wish. El shell tclsh es un shell textual mientras que wish tiene facilidades gráficas.

Tcl evalúa un comando en dos pasos: el análisis (parsing) y la ejecución [25]. En el análisis, la sintaxis de Tcl se usa para guiar al Analizador Tcl en dos subpasos: divide el comando en palabras y realiza sustituciones sin aplicar ningún significado a los valores de las palabras. En el segundo paso se ejecutan los comandos, aplicándose un significado a las palabras del comando. La primer palabra es el nombre del comando y las demas palabras se pasan como argumentos al procedimiento del comando quien las interpreta.

En resumen, cuando Tcl evalúa un comando lo realiza en los siguientes pasos:

1. Análisis del Comando.
   1.1. Agrupamiento. Tcl agrupa varias palabras en un solo argumento en respuesta a los mecanismos de agrupamiento: las comillas dobles y las llaves.
   1.2. Sustitución. Después de agrupar los argumentos, Tcl realiza la sustitución de variables, de comandos y de secuencias de diagonales invertidas.
2. Ejecución del Comando. Después de la sustitución, Tcl usa el nombre del comando para llamar al procedimiento que lo implementa.



3.11. Los Resultados de un Comando.

En resumen, un comando es una cadena de caracteres con campos separados por espacios en blanco. El primer campo es el nombre del comando y los campos restantes son argumentos que se pasan al comando para su interpretación. Si una cadena de caracteres contiene varios comandos, se encuentran separados por líneas nuevas o por el signo ";".

Las comillas dobles permiten agrupar argumentos y sustituir comandos, sustituir variables y sustituir secuencias de diagonales invertidas.

Las llaves también permiten agrupar argumentos pero no permiten ninguna sustitución, exceptuando la sustitución de la secuencia \nl. Las sustituciones solo se procesan una vez y pueden realizarse sobre cualquier campo del comando, incluyendo el nombre y los argumentos.

Cada comando produce dos resultados: un código y una cadena de caracteres. El código indica si el comando tuvo éxito ó no y la cadena proporciona información adicional dependiente del comando. El código que indica si el comando tuvo éxito es TCL_OK, el que indica si hubo algún error es TCL_ERROR según se define en tcl.h. Cuando un comando regresa el código TCL_ERROR, el intérprete Tcl detiene el procesamiento de comandos y regresa el código de error a la entidad que lo llamó. Si hay varias invocaciones al intérprete, anidadas, cada comando anidado regresará el código de error al comando llamador hasta llegar al comando de más alto nivel, el cual enviará el error al usuario.

Es posible capturar con el comando *catch* a las excepciones para manejarlas y evitar que aborte el programa.

3.12. La Instalación de Tcl.

Tcl puede descargarse gratuitamente desde http://www.scriptics.com/software/tcltk. Existen versiones para diferentes plataformas incluyendo a los populares sistemas operativos Linux y Windows.

A la fecha (junio del 2001) la última versión desarrollada es la Tcl 8.4 que se encuentra en su segunda revisión alpha (tcl84a2.exe para Windows). La última versión estable es la Tcl 8.3.3.

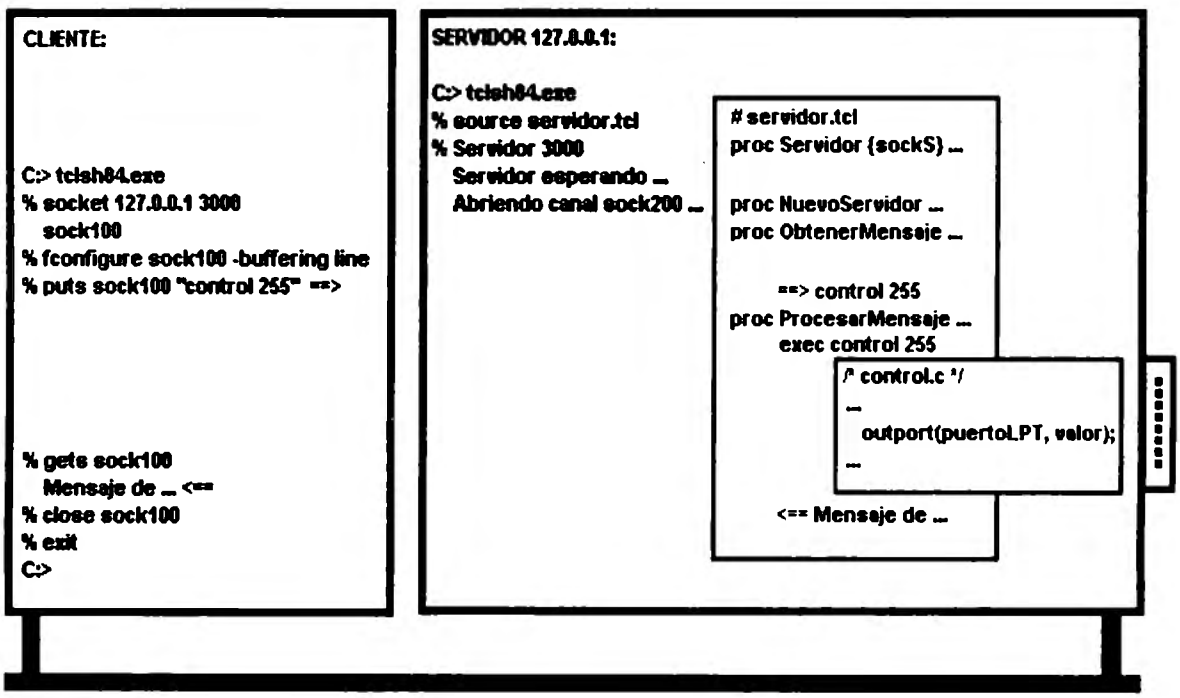
La instalación de Tcl en Windows solo requiere ejecutar el archivo tcl8*.exe para la versión correspondiente, el cual se instala de manera automática en el directorio que se indique. Dentro del directorio de instalación se crea un subdirectorio *bin* el cual contiene a los shells tclsh8?.exe y wish8?.exe. El shell tclsh8?.exe es el shell textual y wish8?.exe es un shell gráfico. Los caracteres "8?" corresponden al número de versión de Tcl.

3.13. El Aprendizaje de Tcl.

El aprendizaje del Lenguaje Tcl es menos complejo en comparación con el aprendizaje de lenguajes como C, C++ o Java, una vez que se han comprendido los principios bajo los cuales funciona. La comunidad Tcl ha generado tutoriales y cursos que permiten introducirse rápidamente al lenguaje [26][27][28][29][30].

3.14. Un Ejemplo de Aplicación de Tcl.

Una de las áreas donde Tcl se ha aplicado de manera existosa es en las aplicaciones que involucran comunicaciones a través de Internet. La siguiente aplicación cliente/servidor desarrollada durante el aprendizaje de Tcl en la presente investigación, permite mostrar como se puede realizar el control remoto de un puerto paralelo de una computadora servidora desde una computadora cliente en una red TCP/IP. La versión que se usó en esta aplicación fue Tcl 8.4 para Windows.



3.14.1. El Servidor de Puerto Paralelo.

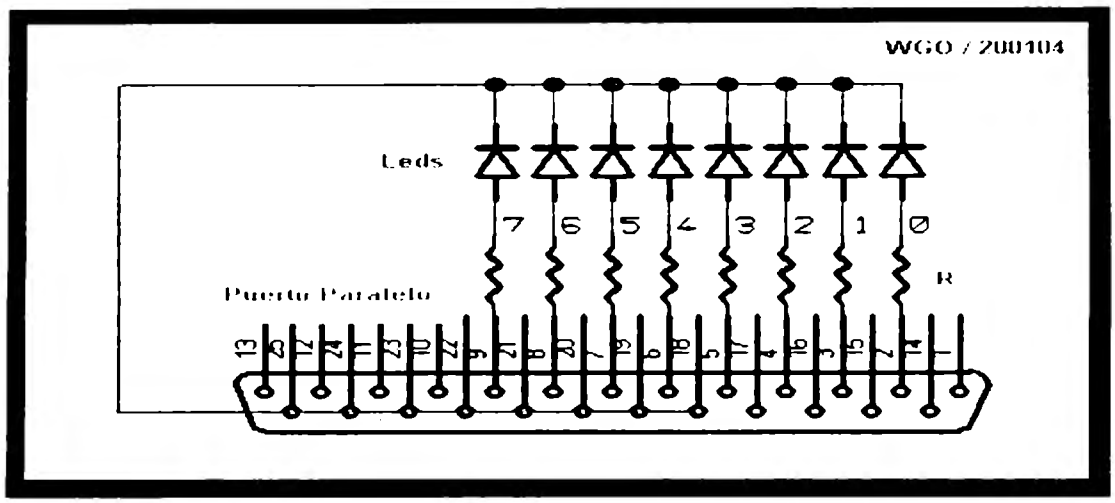
El control del puerto paralelo de una computadora es particularmente interesante porque permite utilizar 8 bits de salida para realizar operaciones que, acopladas a una etapa de potencia, pueden manejar actuadores de diversas características. Además, es prácticamente general el encontrar un puerto paralelo en cada computadora personal.

El programa que controla al puerto paralelo se denomina *control* y espera un argumento entre 0 y 255 que equivale al código binario que se enviará al puerto. Una vez compilado el programa (control.exe) se pueden enviar, localmente, códigos al puerto paralelo.

El programa fuente es el siguiente:

```
/* control.c / wgo / 2001-04 */
/* Compilado con Turbo C++ for Windows, Version 3.1 */

#include <stdio.h>                      /* Requerido por printf */
#include <stdlib.h>                     /* Requerido por atoi   */
#include <dos.h>                        /* Requerido por outport */
#include <string.h>                     /* Requerido por strlen */

int esNumNat(char arregloChar[]) {      /* Verifica si es un número natural: 0, 1, ...*/
  int resultado;
  int longitud;
  int i;
  char c;
  resultado = 1;
  longitud = strlen(arregloChar);
  for (i=0; i<longitud; i++) {
        c = arregloChar[i];
        if ((c < '0') || ('9' < c)) resultado = 0;
  }
  return(resultado);
}

int main(int numArg, char *valArg[]) {
  int valor;
  int puertoLPT = 0x378;

  if (numArg != 2) {
        printf("Numero de argumentos incorrecto (2): %i\n", numArg);
        return(1);
  } else {
        if (!esNumNat(valArg[1])) {
          printf("Valor no numerico natural (0, 1, ...): %s\n", valArg[1]);
          return(2);
        } else {
          valor = atoi(valArg[1]);
          if ((valor < 0) || (255 < valor)) {
                printf("Valor fuera de rango (0, 255): %s\n", valArg[1]);
                return(3);
          } else {
                outport(puertoLPT, valor);
                printf("Operacion Solicitada! Puerto: 0x%x. Valor: %i.\n", puertoLPT, valor);
                return(0);
          }
        }
  }
}
```

Para verificar que el puerto paralelo reciba los códigos, se desarrolló una tarjeta con leds conectados a través de una resistencia de 390 ohms 250 miliwatts, a cada una de las 8 líneas de datos del puerto.



Para poder ser controlado desde otra computadora se requiere de un programa servidor que se encargue de recibir las solicitudes de conexión de los clientes y, una vez establecidas, ordenar la ejecución local del programa de control.

El programa servidor Tcl es el script *servidor.tcl*. En este programa se intenta mostrar algunas de las facilidades de intercomunicación de Tcl, por lo que no se consideraron aspectos relacionados con la seguridad como la autentificación de los clientes y la confiabilidad de los mensajes. El script servidor.tcl define cuatro procedimientos:

```
##########################################################################
# servidor.tcl / wgo / 200104
##########################################################################
```

- Procedimiento Servidor.

  Define un socket *$sockS* para recibir solicitudes de conexión de los clientes. Cuando recibe una solicitud, ordena la ejecución del Procedimiento *NuevoServidor* y le pasa como argumentos el canal del nuevo servidor *canalNS*, la dirección IP del cliente *dirC* y el socket del Cliente *sockC*. Una vez atendida la solicitud de conexión, vuelve a esperar otra solicitud.

```
proc Servidor {sockS} {
  socket -server NuevoServidor $sockS        ;# Llama a NuevoServidor con $canalNS, $dirC,  $sockC
  puts "Servidor esperando conexiones en el socket $sockS ..."
  vwait paraSiempre
}
```

- Procedimiento NuevoServidor.

  Almacena en la variable arreglo *cliente* los datos de la dirección IP *$dirC* y del socket del Cliente *$sockC* usando como índice el canal del nuevo servidor *$canalNS* con la finalidad de llevar un registro de sus conexiones. La variable cliente es global para que pueda ser accesada desde diferentes procedimientos.

  Envía un mensaje a la consola del servidor informando que se tiene un canal de comunicación con el cliente.

  Configura al canal para que el contenido del buffer se envíe cada vez que reciba un fin de línea.

  Finalmente entra en un ciclo que verifica cuando hay información de entrada disponible en el canal. Cuando sucede este evento, ordena la ejecución del procedimiento *ObtenerMensaje*.

  Al procedimiento ObtenerMensaje le envía como argumento el canal del nuevo servidor.

```
proc NuevoServidor {canalNS dirC sockC} {
  global cliente
  set cliente($canalNS) [list $dirC:$sockC]
  puts "Abriendo canal $canalNS con Cliente $cliente($canalNS)"
  fconfigure $canalNS -buffering line
  fileevent $canalNS readable [ObtenerMensaje $canalNS]
}
```

- Procedimiento ObtenerMensaje.

  Este procedimiento se ejecuta cuando en el canal de comunicación hay información de entrada.

  El mensaje se captura en la variable *mensajeEnt* y si existe alguna excepción en su lectura, se registra como tipo de error 1. Si se recibe un fin de archivo se registra como tipo de error 2. Si el mensaje contiene la cadena "fin", significa que el cliente ha enviado su último mensaje y se registra como tipo de error 3.

  Estrictamente solo el tipo de error 1 se puede considerar como tal, pero este mecanismo permite registrar mensajes especiales.

  Si se registra algún tipo de error, se cierra el canal, se elimina la información correspondiente a este canal del registro de conexiones y se envía un mensaje a la consola.

  Si no ocurre algún tipo de error, se envía a la consola del servidor, el mensaje recibido del cliente y se ordena la ejecución del procedimiento *ProcesarMensaje* enviando como argumentos el valor del canal y el mensaje recibido.

```
proc ObtenerMensaje {canalNS} {
  global cliente
  set tipoDeError 0
  if {[catch {gets $canalNS mensajeEnt}]} {set tipoDeError 1}
  if {[eof $canalNS]}                     {set tipoDeError 2}
  if {$mensajeEnt == "fin"}               {set tipoDeError 3}
  if {$tipoDeError != 0} {
    close $canalNS
    puts "Cerrando canal $canalNS con Cliente $cliente($canalNS)"
    unset cliente($canalNS)
  } else {
    puts "Mensaje de $cliente($canalNS):\t$mensajeEnt"
    ProcesarMensaje $canalNS $mensajeEnt
  }
}
```

- Procedimiento ProcesarMensaje.

  Al alcanzar este punto, el servidor tiene un mensaje en *mensajeEnt*, que debe procesar. Verifica primero que el número de argumentos sea correcto. Para esta aplicación se esperan 2 argumentos: el primer argumento es el nombre de un *comando* y el segundo es un argumento para tal comando denominado *valor*.

  Si el comando no es "*control*" ó el valor esta fuera del rango [0, 255], prepara el mensaje de salida indicando el error.

  Si el comando y el valor son apropiados, se ordena la ejecución del comando con el valor correspondiente.

  El resultado regresado por la ejecución del comando sirve para preparar el mensaje de salida. El mensaje de salida se envía a la consola del servidor y al canal que comunica con el cliente.

```
proc ProcesarMensaje {canalNS mensajeEnt} {
  set numArg [llength $mensajeEnt]
  if {$numArg != 2} {
    set mensajeSal "Numero de argumentos erroneo: $numArg"
  } else {
    set comando [lindex $mensajeEnt 0]
    set valor   [lindex $mensajeEnt 1]
    if {$comando != "control"} {
      set mensajeSal "Comando erroneo: $comando"
    } else {
      if {($valor < 0) || (255 < $valor)} {
        set mensajeSal "Valor erroneo: $valor"
      } else {
        set mensajeSal [exec control $valor]
      }
    }
  }
  puts "\t\tRespuesta:\t$mensajeSal"
  puts $canalNS "Mensaje: \"$mensajeEnt\", Respuesta: \"$mensajeSal\""
}
```

En la computadora que funcionará como servidor, se ejecuta el shell tclsh y se carga el archivo que contiene el script con la instrucción source: *% source servidor.tcl.*

Una vez cargado el archivo que contiene el script, se ordena la ejecución del procedimiento *Servidor* indicando el puerto en el que se recibirán las solicitudes de conexión de los clientes. El puerto debe ser superior a 1024 para no entrar en conflicto con los puertos bien conocidos definidos en el conjunto de protocolos TCP/IP y puede llegar hasta 65535.

Si se usa por ejemplo el puerto 3000, la ejecución del procedimiento Servidor se indicaría de la siguiente manera: *Servidor 3000.*

El siguiente listado muestra al script servidor.tcl completo.

```
##############################################################################
# servidor.tcl / wgo / 200104
##############################################################################
proc Servidor {sockS} {
   socket -server NuevoServidor $sockS
   puts "Servidor esperando conexiones en el socket $sockS ..."
   vwait paraSiempre
}

proc NuevoServidor {canalNS dirC sockC} {
   global cliente
   set cliente($canalNS) [list $dirC:$sockC]
   puts "Abriendo canal $canalNS con Cliente $cliente($canalNS)"
   fconfigure $canalNS -buffering line
   fileevent $canalNS readable [ObtenerMensaje $canalNS]
}

proc ObtenerMensaje {canalNS} {
   global cliente
   set tipoDeError 0
   if {[catch {gets $canalNS mensajeEnt}]} {set tipoDeError 1}
   if {[eof $canalNS]}                     {set tipoDeError 2}
   if {$mensajeEnt == "fin"}               {set tipoDeError 3}
   if {$tipoDeError != 0} {
      close $canalNS
      puts "Cerrando canal $canalNS con Cliente $cliente($canalNS)"
      unset cliente($canalNS)
   } else {
      puts "Mensaje de $cliente($canalNS):\t$mensajeEnt"
      ProcesarMensaje $canalNS $mensajeEnt
   }
}

proc ProcesarMensaje {canalNS mensajeEnt} {
   set numArg [llength $mensajeEnt]
   if {$numArg != 2} {
      set mensajeSal "Numero de argumentos erroneo: $numArg"
   } else {
      set comando [lindex $mensajeEnt 0]
```

```
    set valor      [lindex $mensajeEnt 1]
    if {$comando != "control"} {
      set mensajeSal "Comando erroneo: $comando"
    } else {
      if {($valor < 0) || (255 < $valor)} {
        set mensajeSal "Valor erroneo: $valor"
      } else {
        set mensajeSal [exec control $valor]
      }
    }
  }
  puts "\t\tRespuesta:\t$mensajeSal"
  puts $canalNS "Mensaje: \"$mensajeEnt\", Respuesta: \"$mensajeSal\""
}
```

3.14.2. El Cliente.

El cliente es una computadora conectada en red con el servidor usando el conjunto de protocolos TCP/IP. Las operaciones que se deben realizar en el cliente para lograr la comunicación son las siguientes:

1. Ejecutar el shell de Tcl.

   Para ejecutar el shell de Tcl, considerando que se usa la versión Tcl 8.4, se indicaría:

   C:> tclsh84.exe

2. Abrir un canal de comunicación.

   Un canal de comunicación con un servidor se abre usando el comando *socket* e indicando la dirección IP del servidor *dirS* (127.0.0.1 si es la misma computadora que el cliente) y el socket *sockS* donde el servidor espera recibir las solicitudes de conexión. Por ejemplo, si dirS es 127.0.0.1 y sockS es 3000, el comando se indicaría:

   % socket 127.0.0.1 3000

   Tcl respondería con un identificador del canal de comunicación *socknum*. Por ejemplo si socknum es sock100, la respuesta sería:

   ⇒ sock100

   Se pueden abrir más canales de comunicación con el mismo servidor, solicitándolo de la forma antes indicada, con la única diferencia de que la identificación del canal *socknum* que se obtendría sería diferente en cada caso.

3. Configurar el buffer del canal.

   El canal recibe y envía información a través de un buffer con capacidad de algunos kilobytes.

El contenido del buffer se envía al canal generalmente hasta que el buffer se llena. Para enviar el contenido del buffer al canal, cada vez que se reciba una línea en lugar de esperar a que se llene, se debe configurar el buffer del canal con el comando *fconfigure* de la siguiente manera (considerando que la identificación del canal es sock100):

% fconfigure sock100 –buffering line

4. Enviar un mensaje al servidor.

   Para enviar mensajes al servidor se usa el comando *puts* con dos argumentos, la identificación del canal *socknum* y el *mensaje*. Si el mensaje requiere agrupamiento debe enviarse entre comillas dobles:

   % puts sock100 "control 255"

   En este ejemplo, el mensaje es "control 255" el cual será interpretado por el servidor como una orden de control para que envíe al puerto paralelo el código 255, el cual pondrá en un uno lógico a los 8 bits del puerto de salida.

   Otro mensaje podría ser la cadena "fin", el cual sería interpretado por el servidor como una solicitud de cierre del canal. Si el mensaje no tiene una interpretación para el servidor, regresará una respuesta indicándolo.

   % puts sock100 fin

5. Recibir un mensaje del servidor.

   Para recibir los mensajes que el servidor envía en respuesta a los mensajes recibidos, se usa la instrucción *gets* indicando como argumento la identificación del canal *socknum*:

   % gets sock100

   De esta manera los mensajes se reciben en la salida estándar (pantalla), pero usando las instrucciones correspondientes se puede, por ejemplo, capturar la salida en una variable para su posterior procesamiento. Dependiendo del mensaje enviado al servidor será la respuesta recibida.

   Si no se ha enviado ningún mensaje y se solicita recibir un mensaje, el sistema se bloqueará.

6. Cerrar el canal.

   Cuando se requiera cerrar el canal se puede enviar un mensaje "fin" al servidor el cual causará que el servidor cierre su canal. Posteriormente el cliente debería cerrar su canal con el comando *close* y el argumento *socknum*. Tcl puede, sin embargo, cerrar ambos extremos del canal con la única indicación de que se cierre el canal del cliente:

   % close sock100

7. Salir de Tcl.

   Para salir del shell tclsh, se usa el comando *exit*:

   % exit

En resumen, las operaciones que debe realizar el cliente para operar esta aplicación son:

1. Ejecutar el shell de Tcl:	*C:> tclsh84.exe*
2. Abrir un canal de comunicación:	*% socket dirServ sockServ*
Se recibe un identificador de canal:	*⇒ socknum*
3. Configurar el buffer del canal:	*% fconfigure socknum –buffering line*
4. Enviar un mensaje al servidor:	*% puts socknum "mensaje"*
5. Recibir un mensaje del servidor:	*% gets socknum*
6. Cerrar el canal:	*% close socknum*
7. Salir de Tcl:	*% exit*

El estudio del Lenguaje Tcl en esta investigación permitió contar con las facilidades para proceder a integrar una aplicación más compleja a partir de aplicaciones existentes, de una manera más rápida y con menor cantidad de código.

4. El Software SNMP para Administración de Redes.

Para lograr el objetivo fundamental de la investigación, mostrar la posibilidad de que un sistema de administración de redes cuente con el apoyo de un sistema experto, es necesario contar con software que administre la red de computadoras, otro que implemente el sistema experto y finalmente uno que permita integrar las aplicaciones.

Para administrar la red de computadoras existen diversos sistemas usando el protocolo SNMP, tanto comerciales como de dominio público, desarrollados en lenguajes como C++, Java, Perl y Tcl principalmente [31][32]. Los sistemas cuentan con bibliotecas API (Application Program Interface) cuyas funciones SNMP se llaman para realizar las operaciones correspondientes. Algunos incluyen un ambiente que facilita su programación. Una condición fundamental para seleccionar el software de administración de redes es que permita una fácil interacción con otros sistemas.

4.1. Scotty, un Software SNMP en Tcl/Tk.

El sistema administrador seleccionado fue Scotty [33], un producto de dominio público desarrollado en Tcl/Tk por Juergen Schoenwaelder y otros, que puede operar bajo Unix ó Windows.

Entre sus características principales además de las propias de Tcl, están la operación por sesiones, la emisión textual de comandos SNMP, el soporte a la base de datos MIB-2, el soporte a MIB's de compañías particulares, la visualización de los objetos administrables de la base de datos, la representación gráfica de los nodos de la red, el descubrimiento automático de nodos y la capacidad de activar operaciones de supervisión y control.

Este software se ha usado para implementar soluciones de alto nivel en la administración de redes [34]. Scotty consta de dos partes: Tnm y Tkined.

Tnm permite tener acceso a la información de la administración de redes por medio de un API basado en cadenas de caracteres. Puede automatizar interacciones por medio de scripts Tcl. Soporta los protocolos ICMP, DNS, HTTP, SUN RPC, NTP, UDP y SNMP. Las versiones de SNMP implementadas son SNMPv1, SNMPv2c, SNMPv2u y SNMPv3, además de capacidad de acceso a las definiciones MIB.

Tkined está basado en la extensión Tk de Tcl y proporciona un marco de referencia para un sistema de administración de redes por medio de una GUI (Graphic User Interface). Tkined puede supervisar automáticamente una red con enrutadores, conmutadores, concentradores y otros dispositivos que soporten el protocolo SNMP. Tiene la capacidad de crear mapas de red y descubrir redes IP además de otras facilidades para administrar redes.

El software se puede obtener de la página http://www.ibr.cs.tu-bs.de/projects/scotty. En esta investigación se usó la versión Scotty 3.0.0 para Windows

liberada en febrero del 2000. El archivo de instalación se denomina scotty300-snap20000219.exe e incluye Tnm v3.0.0, Tkined v1.5.0 y Tcl/Tk v8.2.3. Para instalarlo se debe ejecutar el archivo de instalación. En el proceso de instalación, se verifica si ya está instalado Tcl/Tk v8.2 y en caso de no estarlo, se propone instalarlo. Por omisión, Windows en español instala Scotty en C:\Archivos de Programa\Scotty y Tcl/Tk en C:\Archivos de Programa\Tcl.

Para ejecutar Tcl\Tk se puede usar la secuencia *Botón de Inicio, Programas, Tcl* y seleccionarse un shell: *tclsh82* ó *wish82*.

Para ejecutar Tkined v1.5.0, se puede usar la secuencia *Botón de Inicio, Programas y Tkined v1.5.0*. Se debe corregir la ubicación del shell wish82.exe en versiones donde se use Archivos de Programa en lugar Program Files.

Para ejecutar Tnm se debe ejecutar un shell de Tcl\Tk y el paquete Tnm según se indica en la siguiente sección.

La documentación sobre Tnm y Tkined está incluída en la instalación.

4.2. Tnm, un Conjunto de Comandos Textuales.

El objetivo de Tnm es simplificar el desarrollo de aplicaciones de administración de redes proporcionando acceso a los protocolos de red por medio de una API basada en caracteres. Esta API simplifica el desarrollo de las aplicaciones porque opera en un nivel muy alto, encapsulando la complejidad de la intercomunicación, además de que el Lenguaje Tcl por su propio enfoque, permite una fácil integración con otros sistemas. En esta investigación se usa la extensión Tnm del Lenguaje Tcl y el modelo derivado del protocolo SNMP para administrar redes.

Para usar Tnm bajo Windows, se siguen los siguientes pasos:

1. Ejecutar un shell de Tcl (tclsh82.exe ó wish82.exe):
   ```
   C\:>...\tclsh82.exe   ó   C\:>...\wish82.exe
   %
   ```

2. Cargar Tnm 3.0 o una versión posterior:
   ```
   % package require Tnm 3.0
   Tnm 3.0.0
   %
   ```

3. Importar el espacio de nombres de Tnm al espacio de trabajo de Tcl para evitar el prefijo Tnm:: en los comandos, por ejemplo snmp ó mib en lugar de Tnm::snmp ó Tnm::mib:
   ```
   % namespace import Tnm::*
   %
   ```

Tnm incluye un conjunto de grupos de comandos, utilerías, variables de ambiente, variables Tcl y archivos de configuración.

Los grupos de comandos que soporta Tnm 3.0 son:

Tnm::dns	Consulta al sistema de nombres de dominio de Internet.
Tnm::icmp	Envía mensajes icmp a los nodos de Internet.
Tnm::ined	Accesa y manipula objetos Tk.
Tnm::job	Invoca procedimientos Tcl a intervalos regulares
Tnm::map	Crea y manipula mapas de red.
Tnm::mib	Carga y consulta definiciones MIB.
Tnm::netdb	Accesa a bases de datos de la red local.
Tnm::ntp	Recupera variables de control de compañeros NTP.
Tnm::snmp	Envía mensajes SNMP a nodos de Internet.
Tnm::sunrpc	Llama y prueba servicios Sun RPC.
Tnm::syslog	Escribe mensajes en un sistema de registro.
Tnm::udp	Envía y recibe datagramas.

Son de interés particular para esta investigación, los grupos de comandos Tnm::snmp y Tnm::mib.

4.3. Los Comandos Tnm::snmp.

Los comandos Tnm::snmp implementan al protocolo SNMP y se han agrupado de la siguiente manera:

Comandos SNMP	Crean sesiones y realizan tareas relacionadas con SNMP.
Comandos Generales de Sesión.	Configuran, consultan, bloquean y destruyen una sesión.
Comandos de la Sesión Generator.	Comandos aplicables en una sesión tipo generator.
Comandos de la Sesión Responder.	Comandos aplicables en una sesión tipo responder.
Comandos de la Sesión Notifier.	Comandos aplicables en una sesión tipo notifier.
Comandos de la Sesión Listener.	Comandos aplicables en una sesión listener.

4.3.1. La Sesión SNMP.

Para realizar operaciones SNMP es necesario abrir una sesión. Una sesión es un objeto que contiene información de control. Se pueden abrir cuatro tipos de sesiones:

Sesión Generator.	Permite enviar comandos get/getnext/bulk/set y esperar respuestas a los mismos, desde un administrador.
Sesión Responder.	Permite recibir comandos get/getnext/bulk/set, procesarlos y regresar mensajes de respuesta, desde un agente.
Sesión Notifier.	Permite enviar notificaciones de eventos, tanto traps que no esperan reconocimiento, como informes que si lo esperan, desde un agente.
Sesión Listener.	Permite recibir notificaciones de eventos, tanto traps que no esperan reconocimiento, como informes que si lo esperan, desde un administrador.

4.3.2. Los Comandos SNMP.

Los comandos para crear sesiones y otras tareas relacionadas con SNMP se denominan comandos SNMP y se muestran en la siguiente lista. Nótese que la primera palabra de estos comandos es precisamente *snmp* y que se ha omitido el prefijo *Tnm::*.

- Comandos SNMP para crear sesiones:
  - snmp generator [option value] — Crea una sesión generator.
  - snmp responder [option value] — Crea una sesión responder.
  - snmp notifier [option value] — Crea una sesión que genera notificaciones.
  - snmp listener [option value] — Crea una sesión que espera notificaciones.

- Comandos SNMP para realizar tareas relacionadas con el protocolo:
  - snmp alias [name [options]] — Crea un alias para una lista de opciones de configuración.
  - snmp delta vbl1 vbl2 — Calcula el incremento entre los elementos de las listas.
  - snmp find [-address addr][-port num][-tag patternList][-type type][-version version] — Regresa identificadores de sesión según opciones.
  - snmp info subject ?pattern? — Regresa información acerca de sujetos como errors, exceptions, pdus, types y versions.
  - snmp oid vbl [index] — Extrae los identificadores de objeto de la lista varbind vbl.
  - snmp type vbl [index] — Extrae el tipo de datos de la lista varbind vbl.
  - snmp wait — Bloquea al sistema hasta que todas las solicitudes asíncronas de todas las sesiones se hayan procesado.
  - snmp watch I/O — Activa o desactiva el desplegado de los paquetes SNMP.

A una solicitud de apertura de cualquier tipo de sesión, el sistema responde con un identificador de sesión, el cual permite orientar los comandos a la sesión correspondiente. Esto es necesario dada la posibilidad de manejar varias sesiones simultáneamente.

Esta investigación requiere realizar operaciones de supervisión y control desde un administrador. Para supervisar, se consulta el valor de las instancias de los objetos administrables y para controlar, se modifica su valor.

La consulta y modificación de las instancias de los objetos administrables o variables, se puede realizar por medio de una sesión tipo "generator".

Para abrir una sesion tipo "generator" se emite el siguiente comando:

```
% snmp generator
```

El sistema responde con un identificador de sesión:

```
snmp#
```

El identificador de sesión *snmp#* será la primer palabra de los comandos que se emitan hacia tal sesión cuando se emitan Comandos Generales de Sesión ó Comandos de la Sesión Generator, Responder, Notifier ó Listener: *snmp# comando [opcion].*

4.3.3. Los Comandos Generales de Sesión.

Una vez abierta una sesión, los comandos que permiten configurarla, consultarla, bloquearla y destruirla se denominan Comandos Generales de Sesión.

- Comandos para configurar y consultar una sesión:
  - snmp# configure [option value ...] — Consulta o cambia el valor de una o varias opciones de configuración.
  - snmp# cget option — Consulta el valor de una opción de configuración.

  Las opciones de configuración de una sesión SNMP son las siguientes:

-address A.B.C.D	Dirección IP de la entidad compañera. El valor por omisión es 127.0.0.1.
-port puerto	Puerto UDP usado por la entidad compañera para recibir mensajes SNMP. El valor por omisión es 161 para las sesiones generator/responder y 162 para las sesiones notifier/listener.
-version snmpversion	Versión usada en la sesión SNMP. Se soportan SNMPv1, SNMPv2c y SNMPv3. El valor por omisión es SNMPv1.
-community comunidad	Cadena que define a la comunidad usada cuando se invocan operaciones SNMP. El valor por omisión es "public".
-timeout segundos	Tiempo en segundos que se espera por una respuesta. El valor por omisión es 5 segundos.
-retries retransmisiones	Número de retransmisiones de una solicitud durante el intervalo timeout. Aplicable en mensajes UDP. El valor por omisión es 3.
-delay milisegundos	Retraso en milisegundos entre mensajes SNMP para evitar congestión en la red. Aplicable en mensajes UDP. El valor por omisión es 0 milisegundos.
-window maxSolAsAct	Número máximo de solicitudes asíncronas activas. Si aumenta el número de solicitudes, Tnm las hace esperar internamente. Aplicable en mensajes UDP. El valor 0 desactiva la opción. El valor por omisión es 10.
-alias nombreAlias	Usa las opciones de configuración contenidas en el alias "nombreAlias", el cual se define con el comando snmp alias.
-tags listaDeEtiquetas	Etiqueta sesiones SNMP para poder agruparlas en relación a un objeto o a una tarea específica. El valor por omisión es la lista vacía {}.

- Comando para bloquear una sesión:
  - snmp# wait [request] — Bloquea al sistema hasta que todas las solicitudes asíncronas de la sesión se hayan procesado.

- Comando para destruir una sesión:
  - snmp# destroy — Destruye una sesión.

Por ejemplo, para consultar la configuración de una sesión SNMP se aplica el comando *configure* a la sesión de interés, por ejemplo *snmp100*, y el sistema responde con la configuración de la sesión:

```
% snmp100 configure
-address 127.0.0.1 -port 161 -version SNMPv1 -community public -transport udp -timeout 5 -
retries 3 -window 10 -delay 0 -tags {}
```

Si se desea reconfigurar la sesión para, por ejemplo, conectarse con el nodo cuya dirección IP es 207.248.165.1, se indicaría:

```
% snmp100 configure -address 207.248.165.1
```

4.3.4. Los Comandos de las Sesiones.

Los comandos de las sesiones generator, responder, notifier y listener se pueden emitir únicamente después de abrirse una sesión y contar con el identificador de sesión snmp# correspondiente.

Comandos de la Sesión Generator:
Los comandos de la sesión generator se usan para implementar administradores SNMP. Un administrador SNMP emite mensajes get, getnext, bulk y set y espera una respuesta a estos mensajes.

• snmp# configure [option value ...]	Consulta o cambia las opciones de configuración de la sesión.
• snmp# bind event [script]	Ejecuta el script cuando ocurre un evento send ó recv. Si se omite el script remueve el enlace.
• snmp# get vbl [script]	Recupera la lista de instancias especificadas en la lista varbind vbl. Si incluye al script opera asíncronamente.
• snmp# getnext vbl [script]	Recupera las instancias lexicográficamente sucesoras de los objetos de la lista varbind vbl. Si incluye al script opera asíncronamente.
• snmp# getbulk nr mr vbl [script]	Implementa recorridos rápidos del arbol MIB.
• snmp# set vbl [script]	Crea y modifica instancias MIB.
• snmp# walk varName vbl body	Recorre un subárbol MIB completo. Versión síncrona.
• snmp# walk vbl script	Recorre un subárbol MIB completo. Versión asíncrona.

Comandos de la Sesión Responder:
Los comandos de la sesión responder se usan para implementar agentes SNMP. Un agente SNMP recibe mensajes get, getnext, bulk y set, los procesa y emite una respuesta.

• snmp# instance label varName [default]	Crea una nueva instancia MIB identificada por label y la asocia a la variable Tcl varName.
• snmp# bind label event [script]	Enlaza un script a una instancia MIB identificada por label. Event define la operación SNMP que arranca la evaluación del script.

Comandos de la Sesión Notifier:
Una sesión notifier se desarrolla en un agente y le permite notificar eventos que ocurren en él, tanto traps que no esperan reconocimiento, como informes que esperan una respuesta como reconocimiento del administrador.

• snmp# bind event [script]	Enlaza scripts a eventos generados tales como eventos send y recv.
• snmp# trap snmpTrapOid vbl	Notifica a un administrador que un evento ha ocurrido. El tipo de evento se define por el identificador de objetos snmpTrapOid como coldStart, warmStart y authenticationFailure. No se espera reconocimiento del administrador a la notificación.
• snmp# inform snmpTrapOid vbl [script]	Informa a un administrador que un evento ha ocurrido. Se espera reconocimiento del administrador al informe.

Comandos de la Sesión Listener:
La sesión listener se desarrolla en un administrador que está preparado para recibir notificaciones tanto traps como informes. Cuando recibe un informe responde con un mensaje de confirmación.

• snmp# bind event [script]	Enlaza scripts a eventos generados tales como eventos send, recv, trap e inform.

4.3.5. La Lista de Variables en SNMP.

El protocolo SNMP opera únicamente sobre instancias o variables MIB. Las instancias o variables MIB son las hojas del árbol Identificador de Objetos. Una variable MIB se describe por medio de *Identificador*, un *Tipo* y su *Valor*. A esta lista se le conoce como *varbind*:

Cada lista varbind se encierra entre llaves cuando contiene varios elementos separados por espacios. En operaciones de recuperación como get y getnext, se puede omitir el tipo y el valor del dato, por lo que se pueden omitir las llaves.

Un mensaje SNMP incluye una lista de listas varbind, denominada *varbindlist*, sobre la que opera. Una lista varbindlist contiene información sobre una o varias variables.

Por ejemplo, la siguiente lista varbindlist contiene información sobre 4 variables, donde las 2 últimas solo incluyen su identificación:

```
{{ Identificador1  Tipo1  Valor1 }
 { Identificador2  Tipo2  Valor2 }
   Identificador3
   Identificador4
}
```

En los mensajes SNMP, El identificador se puede reemplazar parcial o totalmente con una etiqueta ó un nombre que resuelva a un valor del árbol Identificador de Objetos (OID, Object Identifier). Por ejemplo, es posible usar "sysName.0" en lugar del identificador "1.3.6.1.2.1.1.5.0".

Los nombres completamente calificados (FQN, Fully Qualified Name) se forman con el nombre del módulo, un signo de admiración y la etiqueta del nodo: *MODULO!etiqueta*. Si se

usan solo la etiquetas, deben evitarse las repeticiones dentro del módulo. Consúltese el comando Tnm::mib para otras posibles etiquetas.

Tnm siempre regresa una lista varbindlist con los datos completos de las variables, es decir con el identificador, el tipo y el valor de cada variable.

4.3.6. Los Tipos de Datos en SNMP.

Los tipos de datos que soporta el protocolo SNMP son:

• Counter32	Valores en el rango 0 a 2^{32}-1, representados en una cadena conteniendo un número sin signo.
• Counter64	Valores en el rango 0 a 2^{64}-1, representados en una cadena conteniendo un número sin signo.
• Gauge32	Valores en el rango 0 a 2^{32}-1, representados en una cadena conteniendo un número sin signo.
• Integer32	Valores en el rango -2^{31} a 2^{31}-1 representados en una cadena conteniendo un número con signo.
• IpAddress	Dirección IPv4 A.B.C.D.
• TimeTicks	Números naturales que representan el tiempo en centésimas de segundo, módulo 2^{32}, representados en una cadena conteniendo un número sin signo.
• Unsigned32	Valores en el rango 0 a 2^{32}-1, representados en una cadena conteniendo un número sin signo.
• Opaque	Valores con una codificación ASN.1/Ber desconocida. Se incluye para soportar algunas versiones de SNMP que pudieran requerirlo.
• OCTET STRING	Cadena de octetos en hexadecimal (00 a ff) separados por punto y coma.
• OBJECT IDENTIFIER	Secuencia de números naturales desde la raíz del árbol de identificación de objetos hasta un objeto ó instancia MIB representados en una cadena.

4.3.7. Las Operaciones Síncronas ó Bloqueantes.

Scotty tiene la posibilidad de realizar operaciones síncronas y asíncronas, conocidas también como bloqueantes y no bloqueantes respectivamente.

Las operaciones síncronas envían una solicitud y esperan una respuesta. No realizan otra operación hasta que llega la respuesta, es decir se bloquean hasta obtener la respuesta. Tienen un enfoque interactivo.

Por ejemplo, para solicitar el valor de la variable sysName.0 de manera síncrona, se indicaría:

```
% snmp# get sysName.0
```

El sistema se bloquearía hasta obtener la respuesta. Una posible respuesta sería:

```
{1.3.6.1.2.1.1.5.0 {OCTET STRING} WNet1}
```

4.3.8. Las Operaciones Asíncronas ó No Bloqueantes.

Con las operaciones asíncronas o no bloqueantes, se envía la solicitud pero no se espera la respuesta y el sistema solo responde inicialmente con un número de solicitud.

Cuando llega la respuesta, se procesa un script asociado denominado callback. Si la respuesta tarda más de un tiempo determinado, se procesa un timeout. Las operaciones asíncronas tienen un enfoque automático.

El script callback se evalúa a un nivel global y usa secuencias predefinidas %, para tener acceso a la información contenida en la respuesta. Estas secuencias se sustituyen antes de evaluar el script.

La siguiente lista presenta las secuencias % permitidas [35]:

%A	Address.	Se sustituye por la dirección IP del compañero al que se envío el mensaje.
%P	Port.	Se sustituye por el número de puerto del compañero al que se envío el mensaje.
%S	Sesion.	Se sustituye por la identificación de la sesión.
%R	Request.	Se sustituye por la identificación de la solicitud.
%C	Community.	Se sustituye por la cadena de la comunidad en SNMPv1.
%T	Type.	Se sustituye por el tipo de Unidad de Paquete de Datos (PDU).
%E	Error.	Se sustituye por el estatus de error.: noError, noSuchName, ...
%I	Index.	Se sustituye por el índice del elemento de la lista que causó el error.
%V	Variables.	Se sustituye por la lista varbindlist.
%%	%.	Se sustituye por %.

Para mostrar un ejemplo de operación asíncrona, las siguientes instrucciones abren una sesión generator obteniendo el identificador de sesión *snmp0*.

```
C:\>...\tclsh82.exe
% package require Tnm 3.0
Tnm 3.0.0
% namespace import Tnm::*
% snmp generator
snmp0
```

Posteriormente se configura la sesión con la dirección IP 192.168.0.1de un Agente SNMP:

```
% snmp0 configure –address 192.168.0.1
%
```

El ejemplo de operación asíncrona obtendrá el tiempo que el sistema tiene operando consultando a la variable sysUpTime.0. Un script callback denominado "scriptcb" se ejecutará cuando se reciba la respuesta.

Para simplificar la operación, el script scriptcb se prepara en un archivo (scriptcb.tcl), se almacena en el directorio de trabajo de Tcl y se carga con el comando source. A partir de ese momento, el script scritpcb contenido en el archivo se puede ejecutar.

El contenido del archivo scriptcb.tcl es el siguiente:

```
# scriptcb.tcl / wgo / 200104

proc scriptcb {listaDeSecuencias} {
    set V0   [lindex $listaDeSecuencias 0]
    set V1   [lindex $listaDeSecuencias 1]
    set V2   [lindex $listaDeSecuencias 2]
    set V3   [lindex $listaDeSecuencias 3]
    set V4   [lindex $listaDeSecuencias 4]
    set V5   [lindex $listaDeSecuencias 5]
    set V6   [lindex $listaDeSecuencias 6]
    set V7   [lindex $listaDeSecuencias 7]
    set V8   [lindex $listaDeSecuencias 8]

    puts "Dirección IP del Agente:                    $V0"
    puts "Puerto UDP del Agente:                      $V1"
    puts "Identificador de Sesión:                    $V2"
    puts "Número de Solicitud:                        $V3"
    puts "Nombre de la Comunidad:                     $V4"
    puts "Tipo de PDU:                                $V5"
    puts "Estado de Error:                            $V6"
    puts "Indice de la Variable con Error:            $V7"
    puts "Lista de Variables {{Ident Tipo Valor} ... }:  $V8"
}
```

Para cargar el archivo scriptcb.tcl, el cual contiene el script scriptcb, se ejecuta:

```
% source scriptcb.tcl
```

Para consultar asíncronamente a la variable sysUpTime.0 se indica:

```
% snmp0 get sysUpTime.0 "scriptcb {%A %P %S %R %C %T %E %I %V}"
```

El sistema responde con un número de solicitud:

```
27121
```

Cuando se recibe la respuesta a la solicitud, se ejecuta automáticamente *scriptcb* el cual, previa sustitución de las secuencias % recibe sus parámetros (%A ... %V).

La información desplegada por la ejecución del script scriptcb es la siguiente, si el Agente SNMP correspondiente ha sido configurado apropiadamente:

```
    Dirección IP del Agente:                    192.168.0.1
    Puerto UDP del Agente:                      161
    Identificador de Sesión:                    snmp0
    Número de Solicitud:                        27121
    Nombre de la Comunidad:                     public
    Tipo de PDU:                                response
    Estado de Error:                            noError
    Indice de la Variable con Error:            -1
    Lista de Variables {{Ident Tipo Valor} ... }:  1.3.6.1.2.1.1.3.0 TimeTicks 4548020
%
```

Debe comentarse que pruebas desde un administrador en Windows 98, requerían que se ejecutara un comando adicional (por ejemplo % snmp0 configure) para desplegar la respuesta.

4.3.9. La Configuración de Agentes SNMP.

La configuración de los Agentes SNMP permite que sea posible establecer una comunicación con los Administradores SNMP de la red. Los parámetros a configurar en los diferentes agentes son similares, sin embargo es necesario consultar el manual del agente para obtener la información específica de cada configuración.

Un ejemplo de configuración del Agente SNMP en Windows 2000 Server es el siguiente:

1. Ejecutar: Control Panel | Administrative Tools | Services | SNMP Service

2. Configurar:

General.	StartUp Type:	Automatic	
	Service Status:	Started	
Agent.	Contact:	Wenceslao Grande Olguin	
	Location:	Departamento de Redes	
Traps.	Community:	public	
	Destinations:	192.168.0.254	*(Administrador)*
Security.	Accepted Communities:	public	
	Rights:	read write/read only	
	Accepted packets only from:	192.168.0.254	*(Administrador)*

Uno de las familias de enrutadores más populares en la actualidad en nuestro país es la CiscoPro CPA2500 con el Sistema Operativo IOS 11.x. Un ejemplo de configuración del Agente SNMP en esta familia de enrutadores es el siguiente:

1.	Conectarse al enrutador:	telnet router	
2.	Registrarse:	login: nombre pass: clave1	
3.	Activar comandos de configuración:	enable: clave2	
4.	Entrar a modo de configuración:	# config terminal	
5.	Definición de una lista de acceso:	# access-list 1 permit host 192.168.0.254	*(Administrador)*
6.	Definición de una comunidad:	# snmp-server community public ro/rw 1 *(Asociación opcional a una lista de acceso, 1 en el ejemplo)*	
7.	Definición del receptor de traps:	# snmp-server host 192.168.0.254 public	*(Administrador)*
8.	Fin de la configuración:	# <Ctrl> Z	
9.	Almacenamiento de configuración:	# copy run start	

4.4. Los Comandos Tnm::mib.

Otro grupo de comandos de Tnm son los comandos MIB que permiten cargar y consultar definiciones MIB. Por omisión el sistema carga MIB-2. Scotty incluye varios archivos de RFC's de MIB's en el directorio ...\Scotty\lib\tnm3.0.0\mibs.

Cuando se tiene disponible una MIB, se pueden usar nombres en lugar de números OID (Object Identifier) para identificar a las variables MIB. Esto facilita al usuario la identificación de las variables.

Los comandos MIB consultan localmente a los nodos definidos en la MIB, no a las variables instanciadas en el agente, por lo que no se requiere abrir alguna sesión snmp.

- Comandos Tnm::mib:

• mib access node ?varSave?	Regresa el máximo acceso permitido a un nodo y el resultado se puede almacenar opcionalmente en varSave.
• mib children node ?varSave?	Regresa una lista con los subnodos de un nodo.
• mib compare oid1 oid2	Compara los identificadores de objetos oid1 y oid2.
• mib defval node ?varSave?	Regresa el valor por omisión de un nodo.
• mib description nodeOrType ?varSave?	Regresa la descripción textual de un nodo o tipo de datos.
• mib displayhint type ?varSave?	Regresa la sugerencia de desplegado asociada con el tipo de datos.
• mib enums type ?varSave?	Regresa las enumeraciones asociadas con el tipo de datos.
• mib exists nodeOrType	Prueba si el nodo ó el tipo de datos existen en MIB.
• mib file nodeOrType	Regresa el nombre del archivo que contiene la definición del nodo ó tipo de datos.
• mib format nodeOrType value	Aplica las reglas de formato asociadas con un nodo ó tipo de datos, a un valor.
• mib index node	Regresa la lista de etiquetas usadas como llave en las tablas.
• mib info subject ?pattern?	Regresa información del sujeto indicado. El patrón opcional se usa para seleccionar un subconjunto de la información
• mib label nodeOrType	Regresa la etiqueta de un nodo ó tipo de datos.
• mib length oid	Regresa la longitud del identificador de objetos oid. La longitud del identificador se define como el número de subidentificadores.
• mib load file	Carga las definiciones MIB contenidas en file. El analizador interno lee el archivo y crea estructuras de datos apropiadas en la memoria principal.
• mib macro nodeOrType	Regresa el nombre de la macro ASN.1 usada para definir el nodo ó tipo de datos.
• mib member node	Regresa la lista de miembros de un grupo de objetos ó de un grupo de notificación, identificados por node.
• mib module nodeOrType	Regresa el nombre del módulo que contiene la definición del nodo ó tipo de datos.
• mib name node	Regresa el nombre totalmente calificado (FQN) asociado con un nodo del árbol OID.
• mib oid node	Regresa el identificador de objetos (oid) del nodo.
• mib pack oid value ?value ...?	Regresa el identificador de objetos completo, necesario para tener acceso a un elemento de la tabla. El argumento oid identifica un renglón de la tabla y value la instancia.
• mib parent node	Regresa el nodo padre del nodo.
• mib range type	Regresa las restricciones de rango asociadas con el tipo de datos.
• mib scan nodeOrType value	Convierte un valor desplegado en su representaciónn original.
• mib size type	Regresa las restricciones de tamaño asociadas con el tipo de datos.
• mib split node	Regresa una lista donde el primer elemento es el identificador de objeto para el tipo de objeto y el segundo elemento contiene el identificador de la instancia.
• mib status nodeOrType	Regresa el estado de la definición del nodo ó tipo de datos: "current", "deprecated" u "obsolete".

• mib subtree oid1 oid2	Compara dos identificadores de objetos oid1 y oid2. Si oid2 está contenido en oid1 regresa un 1, en otro caso regresa un 0.
• mib syntax nodeOrType	Regresa la sintaxis ASN.1 asociada con el nodo ó tipo de datos.
• mib type node	Regresa el tipo de datos asociado al nodo.
• mib unpack oid	Extrae la identificación de la instancia del identificador de objetos oid que identifica un elemento en una tabla.
• mib variables node	Regresa las variables que deben estar presentes en una notificación identificada por el nodo.
• mib walk varName node body	Evalúa el cuerpo del script "body" para cada nodo en el subárbol OID con raíz en node. La variable varName almacena al identificador de objeto del nodo actual cuando se evalúa el script.

Para poder ejecutar los comandos del grupo mib, es necesario ejecutar un shell de Tcl, cargar Tnm e importar el espacio de nombres de Tnm.

4.5. Ejemplos de Aplicación de Tnm.

4.5.1. El Arbol de Identificación de Objetos OID.

El Arbol de Identificación de Objetos OID es objeto de referencias constantes en esta investigación. Dada el número considerable de nodos en SNMP, se desarrolló un script que desplegara nodos del Arbol OID.

La ejecución del script *arbolOID.tcl* permite desplegar los nodos incluyendo sus etiquetas. El desplegado inicia en la raíz y puede desplegar todos los nodos o hasta un nodo máximo. La indicación del nodo máximo (nodoMax), se indica dentro del script. El desplegado completo del Arbol OID puede tardar varios minutos.

El script arbolOID.tcl se ejecuta usando el comando source, el cual carga y ejecuta el script. Cuando se solicita un desplegado hasta un nodo determinado del árbol OID, se obtiene lo siguiente (obsérvese que se despliegan nodos, no instancias):

```
% source arbolOID.tcl

Arbol OID con nodos previos a: 1.3.6.1.2.1.1.9:

1                           iso
1.3                         iso.org
1.3.6                       iso.org.dod
1.3.6.1                     iso.org.dod.internet
1.3.6.1.1                   iso.org.dod.internet.directory
1.3.6.1.2                   iso.org.dod.internet.mgmt
1.3.6.1.2.1                 iso.org.dod.internet.mgmt.mib-2
1.3.6.1.2.1.1               iso.org.dod.internet.mgmt.mib-2.system
1.3.6.1.2.1.1.1             iso.org.dod.internet.mgmt.mib-2.system.sysDescr
1.3.6.1.2.1.1.2             iso.org.dod.internet.mgmt.mib-2.system.sysObjectID
1.3.6.1.2.1.1.3             iso.org.dod.internet.mgmt.mib-2.system.sysUpTime
1.3.6.1.2.1.1.4             iso.org.dod.internet.mgmt.mib-2.system.sysContact
1.3.6.1.2.1.1.5             iso.org.dod.internet.mgmt.mib-2.system.sysName
1.3.6.1.2.1.1.6             iso.org.dod.internet.mgmt.mib-2.system.sysLocation
1.3.6.1.2.1.1.7             iso.org.dod.internet.mgmt.mib-2.system.sysServices
1.3.6.1.2.1.1.8             iso.org.dod.internet.mgmt.mib-2.system.sysORLastChange

invoked "break" outside of a loop
%
```

El script arbolOID.tcl es el siguiente:

```
# arbolOID.tcl / wgo / Despliega el Arbol OID hasta nodoMax

proc esUltimoNodo {nodoHijo} {
    global nodoMax
    if {[string compare $nodoHijo $nodoMax] >= 0} {
        return 1
    } else {
        return 0
    }
}
proc listarNodos {listaHijos etiqPadre} {
    global nodoMax
    if {[llength $listaHijos] == 0} {return}                    ;# 0 Hijos
    if {[llength $listaHijos] == 1} {                           ;# 1 Hijo
        set hijo $listaHijos
        if {![esUltimoNodo $hijo]} {                            ;# Ultimo: No
            set etiqHijo "$etiqPadre.[mib label $hijo]"
            puts [format "%-25s %-25s" $hijo $etiqHijo]
            set listaNietos [mib children $hijo]
            listarNodos $listaNietos $etiqHijo
        } else {                                                ;# Ultimo: Si
            break
        }
        return
    }
    foreach hijo $listaHijos {listarNodos $hijo $etiqPadre}     ;# >1 Hijo
}

# set nodoMax "Ultimo"                                          ;# Todos los nodos
set nodoMax "1.3.6.1.2.1.1.9"                                   ;# Hasta nodoMax
puts "\nArbol OID con nodos previos a: $nodoMax:\n"

set oidRaiz    "1."                                             ;# Inicialización
set etiqRaiz   "iso"
puts [format "%-25s %-25s" "1" $etiqRaiz]
set listaHijosRaiz [mib children $oidRaiz]

listarNodos $listaHijosRaiz $etiqRaiz                           ;# Inicia listado
```

Para listar todos los nodos indíquese un valor cadena para nodoMax, superior lexicográficamente a cualquier nodo posible, por ejemplo "Ultimo". Un detalle interesante del script es que se desarrolló usando las facilidades de recursión de Tcl.

4.5.2. Los Nodos MIB con Acceso Read-Write.

Para desarrollar la investigación se requería determinar que variables de MIB-2 podían ser modificadas, con la finalidad de mostrar la posibilidad de controlarlas. Tales variables debían tener acceso "read-write". Como las variables heredan el tipo de acceso de su nodo padre, para conocer tales variables, era suficiente saber si su nodo padre tenía acceso "read-write".

Para desplegar a los nodos de MIB-2 con acceso read-write, se desarrolló el script denominado *nodosRW.tcl*. Este script se ejecuta con un argumento que indica el número máximo de nodos a desplegar.

Por ejemplo, para desplegar un máximo de 10 nodos con acceso read-write, se ejecuta la siguiente secuencia de comandos:

```
%
% package require Tnm 3.0
Tnm 3.0.0
% namespace import Tnm::*
% source nodosRW.tcl
% nodosRW 10
```

La respuesta del script es la siguiente:

```
% nodosRW 10

Nodos de MIB-2 con Acceso read-write (maximo: 10):

   1: OID: 1.3.6.1.2.1.1.4          Nombre: SNMPv2-MIB!sysContact
   2: OID: 1.3.6.1.2.1.1.5          Nombre: SNMPv2-MIB!sysName
   3: OID: 1.3.6.1.2.1.1.6          Nombre: SNMPv2-MIB!sysLocation
   4: OID: 1.3.6.1.2.1.2.2.1.7      Nombre: IF-MIB!ifAdminStatus
   5: OID: 1.3.6.1.2.1.3.1.1.1      Nombre: RFC1213-MIB!atIfIndex
   6: OID: 1.3.6.1.2.1.3.1.1.2      Nombre: RFC1213-MIB!atPhysAddress
   7: OID: 1.3.6.1.2.1.3.1.1.3      Nombre: RFC1213-MIB!atNetAddress
   8: OID: 1.3.6.1.2.1.4.1          Nombre: IP-MIB!ipForwarding
   9: OID: 1.3.6.1.2.1.4.2          Nombre: IP-MIB!ipDefaultTTL
  10: OID: 1.3.6.1.2.1.4.21.1.1     Nombre: RFC1213-MIB!ipRouteDest

Numero de Nodos desplegados: 10

%
```

Para operar sobre las variables de interés, por ejemplo para consultar su valor, se ejecutarían los comandos snmp correspondientes indicando la instancia respectiva, por ejemplo:

```
% snmp# get sysName.0
{1.3.6.1.2.1.1.5.0 {OCTET STRING} WNet1}
% snmp# get ifAdminStatus.1
{1.3.6.1.2.1.2.2.1.7.1 Integer32 up}
```

El listado del script nodosRW.tcl es el siguiente:

```
# nodosRW.tcl / wgo / Despliega los Nodos de MIB-2 con Acceso read-write.

proc nodosRW {maxNodos} {
   set numNodos 0                                    ;# Contador de Nodos
   puts "\nNodos de MIB-2 con Acceso read-write (maximo: $maxNodos):\n"

   mib walk oidNodo mib-2 {                          ;# Evalua {...} para cada Nodo de mib-2
      if {($numNodos < $maxNodos)} {                 ;# Si no se ha llegado a maxNodos ...
         set acceso [mib access $oidNodo]            ;# Obtiene Acceso de cada Nodo
         if {($acceso == "read-write")} {            ;# Si el Acceso del Nodo es read-write ...
            incr numNodos                            ;# Incrementa Contador de Nodos
            set nomNodo [mib name $oidNodo]          ;# Nombre del Nodo
                                                     ;# Despliega OID y Nombre del Nodo
            puts [format "%4s: OID: %-25s Nombre: %-25s" $numNodos $oidNodo $nomNodo]
         }
      }
   }

   puts "\nNumero de Nodos desplegados: $numNodos\n"
}
```

En este capítulo se analizaron los grupos de comandos Tnm::snmp y Tnm::mib de Scotty, los cuales se pueden usar como herramientas para administrar de manera textual a una red SNMP.

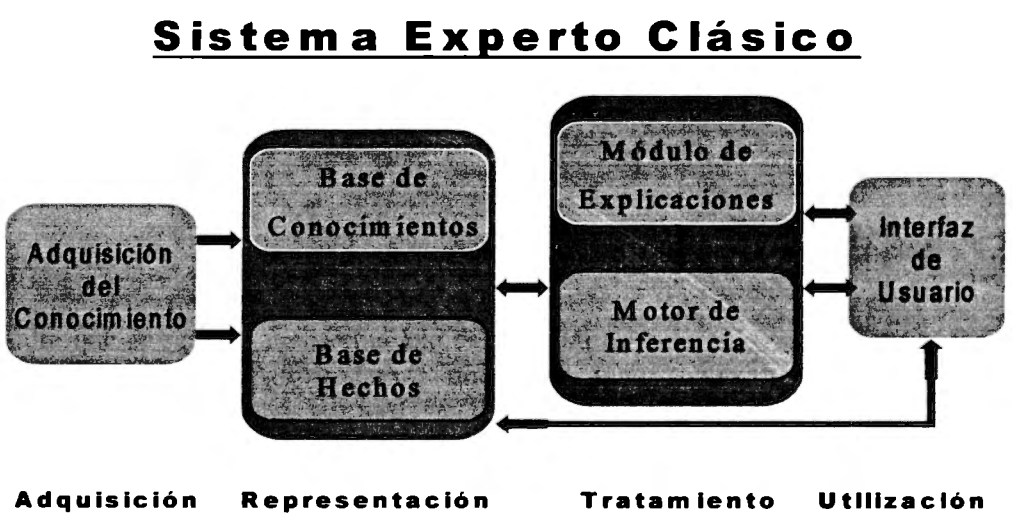
5. La Implementación del Sistema Experto Inicial.

5.1. La Arquitectura del Sistema Experto.

Un Sistema Experto proporciona experiencia en un dominio del conocimiento. Esta rama de la Inteligencia Artificial se ha enfocado a tareas complejas y repetitivas que operan pasivamente al esperar una solicitud para enviar una respuesta y normalmente no son adaptables, no modifican su comportamiento de manera automática al modificarse su entorno. Tienen similitudes con los Agentes de Software, pero estos se enfocan a tareas no complejas, realizan diferentes acciones, son activos y adaptables.

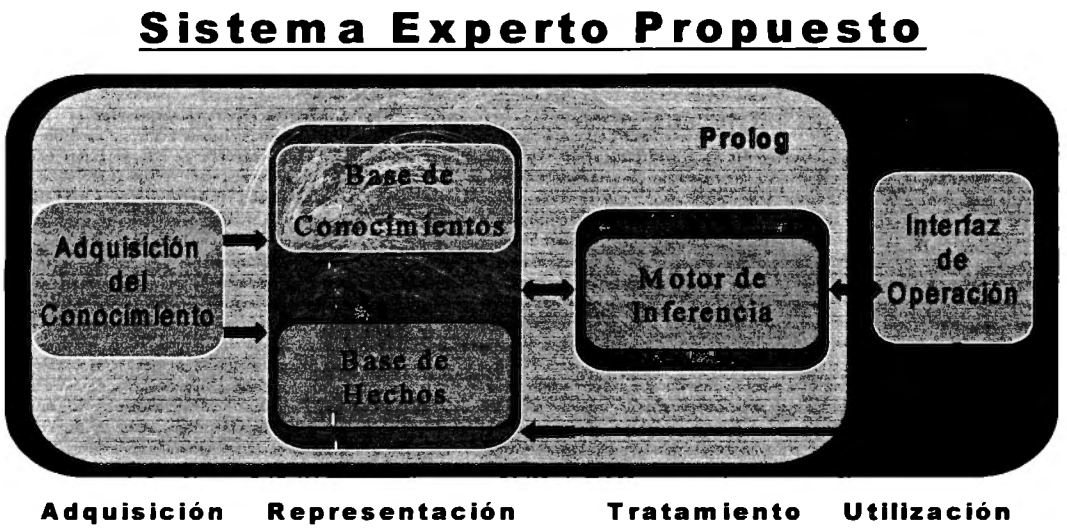
Un Sistema Experto codifica la experiencia de uno ó varios expertos en una Base de Conocimientos. Esta Base de Conocimientos puede enriquecerse conforme se le introducen codificaciones adicionales que representen nuevas experiencias en el dominio.

Para interactuar con el sistema, el usuario tradicionalmente lo realiza de manera interactiva por medio de una Interfaz de Usuario. El usuario proporciona una Base de Hechos ó características particulares de su problema la cual, junto con la Base de Conocimientos, se usa para obtener una respuesta. El mecanismo para obtener una respuesta se basa en un Motor de Inferencia que deduce recomendaciones a partir de la Base de Conocimientos y de la Base de Hechos. La figura siguiente muestra la arquitectura de un Sistema Experto Clásico [5]:



En un Sistema Experto Clásico, el Módulo de Adquisición del Conocimiento es el mecanismo por el que se alimenta la Base de Conocimientos y ocasionalmente la Base de Hechos en la arquitectura tradicional, lo que permite una gran flexibilidad para operar en diferentes ambientes. Por ejemplo, es posible adquirir información de sensores ó de bases de datos en tiempo real. En esta arquitectura, el Interfaz de Usuario permite la consulta interactiva y normalmente alimenta a la Base de Hechos.

La arquitectura propuesta para esta investigación difiere de la arquitectura clásica. La arquitectura clásica opera interactivamente, la arquitectura propuesta opera automáticamente a través de una Interfaz de Operación.



En la arquitectura propuesta, el Módulo de Adquisición del Conocimiento es básicamente el proceso de codificación de la Base de Conocimientos y parcialmente de la Base de Hechos. Dado su enfoque de operación automática, la arquitectura propuesta no considera un Módulo de Explicaciones.

La arquitectura propuesta consta de las siguientes partes:

- Adquisición del Conocimiento.
  - Módulo de Adquisición del Conocimiento
- Representación del Conocimiento
  - Base de Conocimientos
  - Base de Hechos
- Tratamiento del Conocimiento
  - Motor de Inferencia
- Utilización del Conocimiento
  - Interfaz de Operación

5.2. La Representación del Conocimiento Experto.

En los Sistemas Expertos, la representación del conocimiento se materializa en la Base de Conocimientos y en la Base de Hechos. El tratamiento de los conocimientos se realiza por medio del Motor de Inferencia. La Base de Conocimientos es la representación de un tipo de conocimiento conocido como conocimiento heurístico el cual se deriva de la experiencia de

los expertos en el dominio de conocimiento en cuestión. El tipo de conocimiento heurístico es un conocimiento empírico ó reglas de dedo que usan los expertos para resolver rápidamente los problemas que se les presentan y que deducen en el ejercicio de su profesión.

La forma básica en que se representa el conocimiento es por medio de reglas de inferencia conocidas como *modus ponendo ponens, Si Entonces ó If Then.* Estas reglas son la estructura básica del conocimiento. Las reglas SI ENTONCES relacionan premisas para inferir conclusiones. Las premisas y las conclusiones de estas reglas operan sobre datos.

Para representar a los datos de manera que puedan usarse con las reglas de inferencia, se usan varios esquemas [36]:

- Pares Atributo-Valor.
  Aplicable a un solo tipo de objeto.
- Tripletas Objeto-Atributo-Valor.
  Aplicable a varios tipos de objetos.
- Registros Objeto, Atributo_1, Atributo_2, ..., Atributo_n.
  Aplicable a varios tipos de objetos con varios atributos.
- Marcos.
  Aplicable a varios tipos de objetos con varios atributos, incluyendo acciones inteligentes y herencia entre objetos.

En esta investigación se usó la representación de datos atributo-valor, suficiente para lograr el objetivo de la investigación, aún cuando se pueden usar esquemas de representación de datos más complejos para situaciones que así lo requieran.

5.3. Las Técnicas de Inferencia.

Las técnicas que se usan en el Motor de Inferencia para deducir las conclusiones son de dos tipos: las que son guiadas por datos ó con encadenamiento hacia delante (forward chaining) y las que son guiadas por metas ó con encadenamiento hacia atrás (backward chaining).

Las técnicas de inferencia guiadas por datos usan las reglas SI ENTONCES y a partir de los datos iniciales (parte SI), deducen una solución (parte ENTONCES). Esta técnica tiende a generar todas las soluciones posibles.

Las técnicas de inferencia guiadas por metas también usan las reglas SI ENTONCES pero en sentido inverso. Tratan de verificar solo una posible solución fraccionando repetitivamente la meta (ENTONCES) en submetas (SI) hasta que las submetas pueden probarse a partir de las premisas del problema.

El encadenamiento hacia atrás se concentra en verificar solo una solución y tiene la particularidad de solicitar datos solo cuando los necesita. Si se requiere verificar una solución adicional, se hace uso de un mecanismo conocido como retroceso (backtracking).

Tanto las técnicas de inferencia guiadas por datos como las técnicas guiadas por metas se pueden usar para implementar un Motor de Inferencia.

5.4. El Lenguaje Prolog.

El lenguaje Prolog [37][38][39][40] ha sido tradicionalmente el lenguaje seleccionado para desarrollar aplicaciones relacionadas con la Inteligencia Artificial. Prolog tiene un Motor de Inferencia guiado por metas ó con encadenamiento hacia atrás y se basa en reglas SI ENTONCES. Este fué el lenguaje seleccionado para la codificación del conocimiento experto en esta investigación.

En Prolog, las reglas se pueden escribir de la siguiente manera:

cabeza (parte ENTONCES) :- cuerpo (parte SI).

Los hechos se pueden considerar Reglas sin condiciones:

cabeza (parte ENTONCES).

Para ejemplificar la estructura de un programa en Prolog, considere el código Prolog de una Base de Conocimiento almacenado en el archivo *programa.pro*:

```
% programa.pro

% Base de Conocimiento
c(3) :- a(1), b(2).                    % SI a=1 Y b=2 ENTONCES c=3
d(4) :- c(3).                          % SI c=3       ENTONCES d=4
```

La coma en la parte del cuerpo (parte SI) para separar términos, representa al operador de conjunción (Y); dos reglas con la misma cabeza implican al operador de disyunción (O) y la palabra *not* representa al operador de negación. Existen diversos operadores que se pueden aplicar en el lenguaje Prolog además de la posibilidad de definir otros.

Una vez preparado el archivo, se ejecuta Prolog y se carga el programa. En la versión Amzi Prolog, se tiene:

```
Amzi! Prolog Listener
?- consult('c:\\programa.pro').
yes
```

Para incluir los hechos particulares (Base de Hechos) relacionados con un problema específico se puede usar el comando asserta:

```
?- asserta(a(1)).                      % Incluye al hecho a(1)
yes
?- asserta(b(2)).                      % Incluye al hecho b(2)
yes
```

Cuando se han proporcionado las reglas (Base de Conocimientos) y los hechos (Base de Hechos), se pueden realizar consultas.

La consulta en Prolog opera en base a un proceso de unificación y resolución. Por medio de la unificación, el Motor de Inferencia de Prolog determina si hay un unificador que iguale la estructura de un término de la consulta con la cabeza de una regla (parte ENTONCES).

Si existe la unificación, se procede a la resolución reemplazando el término unificado de la consulta con el cuerpo de la regla (parte SI) ó con nada si es un hecho. Este proceso se repite hasta que la consulta está vacía y el unificador final es la solución ó hasta que se determina que falla una unificación y la consulta no puede satisfacerse.

Si se ha tenido éxito, es posible que existan más posibilidades de unificación las cuales pueden obtenerse por medio del mecanismo de retroceso (backtracking), que en Prolog se invoca con el signo ";".

Por ejemplo, para consultar si el atributo *d* de un objeto, vale *X* en el programa Prolog anterior, se invocarían las siguientes operaciones:

```
?- d(X).                          % El atributo d vale X ?
X = 4                             % Si, con el unificador X = 4
      ;                           % Alguna otra ";" solución ?
no                                % No
?-
```

5.5. El Sistema Experto Inicial.

La implementación del Sistema Experto Inicial permite mostrar la forma en que se codificarían las experiencias de los expertos en redes. Este Sistema Experto Inicial se enriquecería progresivamente con la inclusión de nuevas experiencias y la afinación de las ya incluídas.

El Sistema se desarrolló en Prolog. La consulta interactiva del sistema se puede realizar directamente desde este entorno. Para poder realizar consultas automáticas al Sistema Experto, su implementación en Prolog se incrustó en una Interfaz de Operación desarrollada en lenguaje C [36], según se muestra en el siguiente Capítulo.

La Base de Conocimientos desarrollada para esta investigación consideró hacer uso de instancias de nodos MIB con acceso read-write que, siguiendo la filosofía de SNMP, no causaran un daño grave si ocurría algún error en su operación. El programa Prolog, usa un esquema de representación de datos atributo-valor en la Base de Conocimientos y en la Base de Hechos.

Para implementar al Sistema Experto Inicial, se usó la versión gratuita de Amzi Prolog [40], la cual se puede descargar de http://www.amzi.com/download/freedist.htm .

La Base de Conocimientos del Sistema Experto Inicial desarrollada para esta investigación, codifica un criterio para recomendar si se activa (up) el estado administrativo "ifAdminStatus" de una interfaz dependiendo de ciertas condiciones de operación del propio interfaz.

Como Sistema Experto Inicial, este sistema representa solo un ejemplo ilustrativo del conocimiento experto en este dominio. Los nombres de las variables -atributos-, se tomaron de sus identificaciones en MIB-2.

El código Prolog del Sistema Experto Inicial es el siguiente:

```
% experto.pro
% Esquema de representacion de datos: atributo(valor)

% Base de Conocimientos:
ifAdminStatus(up) :-  ifOperStatus(up),
                      ifOutErrors(V0), maxOutErrors(V0M), V0 =< V0M,
                      ifInErrors(V1), maxInErrors(V1M), V1 =< V1M,
                      ifInUnknownProtos(V2), maxInUnknownProtos(V2M),
                      V2 =< V2M,
                      sysContact(V3), authContact(V3).

% Base de Hechos (Parte Estática)
maxOutErrors(1).
maxInErrors(5).
maxInUnknownProtos(200).
authContact(beatrizTrejoMonroy).
authContact(wenceslaoGrandeOlguin).
```

Las reglas de inferencia generan la conclusión de que el estado administrativo "ifAdminStatus" de la interfaz debe activarse (up) si:

- El estado operativo de la interfaz indica que está activo,
- El número de errores de salida no excede el valor máximo permitido,
- El número de errores de entrada no excede el valor máximo permitido,
- El número de mensajes recibidos con protocolos desconocidos no excede el valor máximo permitido y
- El contacto del sistema iguala a uno de los contactos autorizados.

La Base de Hechos es el conjunto de información que representa un problema determinado. El Motor de Inferencia usa estos hechos junto con las reglas de la Base de Conocimientos para inferir conclusiones.

Para disminuir la cantidad de información en tráfico, una parte de la Base de Hechos que se ha denominado la Parte Estática, se ha incluido en el programa experto.pro. Estos hechos son prácticamente comunes a todas las consultas y representan los valores máximos permitidos para errores y los contactos autorizados.

Los hechos adicionales que se requieren para generar una conclusión, forman la Parte Dinámica y se deben proporcionar al solicitarse una consulta.

Los siguientes pares atributo(valor) integran la Parte Dinámica de la Base de Hechos:

```
ifOperStatus(Valor)
ifOutErrors(Valor)
ifInErrors(Valor)
ifUnknownProtos(Valor)
sysContact(Valor)
```

5.6. La Prueba Interactiva del Sistema Experto Inicial.

Una vez preparado el programa, se ejecuta Prolog y se carga el programa (experto.pro) para poder consultarse:

```
?- consult('C:\\PrologAmzi\\a5-0-18\\experto.pro').     % Se carga el programa experto.pro
yes                                                     % Respuesta de Prolog
```

Para la presente investigación, la Parte Dinámica de la Base de Hechos relacionada con una consulta en particular podría ser la siguiente:

```
                                                        % Base de Hechos (Parte Dinámica)
?- asserta(ifOperStatus(up)).                           % Hecho ifOperStatus(up).
yes

?- asserta(ifOutErrors(0)).                             % Hecho ifOutErrors(0)
yes

?- asserta(ifInErrors(3)).                              % Hecho ifInErrors(3)
yes

?- asserta(ifInUnknownProtos(100)).                     % Hecho ifInUnknownProtos(100)
yes

?- asserta(sysContact(wenceslaoGrandeOlguin)).          % Hecho sysContact(wen...)
yes
```

La consulta que se realiza al Sistema Experto Inicial es saber si se debe activar (up) o no, el estado administrativo "ifAdminStatus" que debe tener una interfaz, dados los hechos proporcionados.

Al procesarse la consulta, se puede obtener como solución al unificador "up" si se debe activar el estado administrativo de la interfaz ó se puede obtener como respuesta "no" si falla la unificación.

Después de haberse encontrado una solución, se puede solicitar por medio del signo ";" que se busquen soluciones adicionales por medio del mecanismo de retroceso.

Como para este ejemplo solo existe una solución dada por la unificación "X = up", cuando se soliciten más soluciones, el sistema debe responder "no".

Un ejemplo de consulta interactiva al Sistema Experto Inicial es la siguiente:

```
?- ifAdminStatus(X).                % Consulta si ifAdminStatus vale X
X = up                              % Si, con el unificador X = up
        ;                           % Alguna otra ";" solución ?
no                                  % No
?-
```

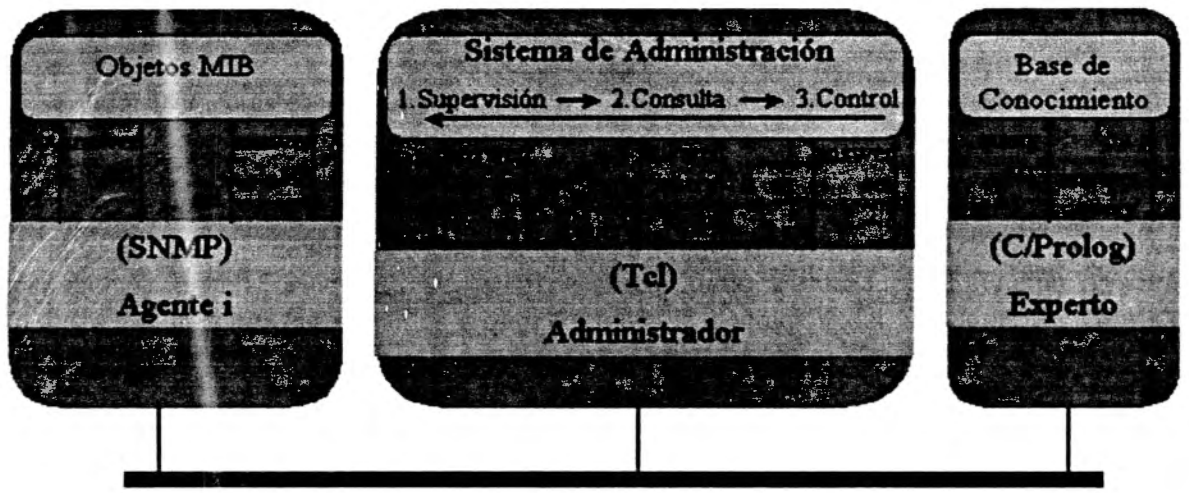
Una vez probada la operación del Sistema Experto Inicial de manera interactiva, en el siguiente capítulo se implementa su interacción automática con un Sistema Administrador de Redes a través de una Interfaz de Operación.

6. Administración Automática de Agentes SNMP con Soporte Experto.

6.1. El Modelo Agente-Administrador-Experto.

En este capítulo se muestra la posibilidad de proporcionar soporte experto a un sistema de administración de redes de manera automática. La interacción entre el Agente en el nodo administrable, el Administrador y el Experto, se propone bajo un Modelo Agente-Administrador-Experto según se muestra en la siguiente figura.

Modelo Agente-Administrador-Experto



El Modelo Agente-Administrador-Experto muestra que el Administrador interacciona con un Agente i y con un Experto en Ciclos de Supervisión, Consulta y Control (CSCC). Los ciclos CSCC a desarrollar con la Lista de Agentes así como la propia Lista de Agentes, pueden definirse según la problemática particular de cada red.

Para un Agente determinado, un ciclo CSCC consta de 3 fases:

- Fase de Supervisión. Esta es la primer fase del ciclo de Supervisión, Consulta y Control. El Sistema de Administración inicia un ciclo solicitando al Agente, información sobre el valor de las variables MIB que le permitan al Administrador supervisar la operación del Agente. Esta fase se realiza con operaciones get y getnext del Protocolo SNMP.
- Fase de Consulta. Con la información proporcionada en la Fase de Supervisión, el Sistema de Administración realiza una consulta al Sistema Experto. El Sistema Experto responde con la información necesaria para ejecutar las acciones de control requeridas sobre

el Agente. Esta fase se realiza a través del Interfaz de Operación que tiene incrustado al programa Experto.

- Fase de Control. Con la información proporcionada por el Sistema Experto, el Sistema de Administración implementa las acciones de control necesarias sobre el Agente, las cuales consisten en modificaciones a los valores de variables MIB. Esta fase se realiza con operaciones set de SNMP.

6.2. El Sistema de Administración.

El control de las operaciones se lleva a cabo por el Sistema de Administración. Para facilitar su interacción con el software SNMP del Agente y su interacción con el ejecutable del Interfaz de Operación del Sistema Experto, se usó el Lenguaje Tcl por sus facilidades integradoras.

El script en Tcl que realiza las funciones administrativas define primero un procedimiento de supervisión denominado supervisarAgente el cual se usa en la Fase de Supervisión. Posteriormente se carga la extensión Tnm para poder realizar operaciones SNMP con el Agente y dependiendo de las particularidades de la red, se proporciona un número de ciclos a desarrollar y una lista de las direcciones IP de los Agentes a administrar.

Por cada ciclo y por cada Agente en la lista, se realizan las siguientes operaciones:

- Supervisión.
  Se abre una sesión con el Agente y se le supervisa con el procedimiento supervisarAgente para obtener los valores de las variables MIB de interés, incluyendo la instancia LAN de las variables asociadas a un Interfaz de Red.
- Consulta.
  Se consulta al Sistema Experto ejecutando el programa InterfazAut con los valores obtenidos de la supervisión.
- Control.
  Dependiendo del resultado de la consulta, se realizan acciones de control sobre el Agente, por medio de la modificación de sus variables MIB.

Antes de continuar con el siguiente Agente, el script Administrador.tcl destruye la sesión y espera un tiempo para no incrementar considerablemente el tráfico administrativo en la red.

Cuando se realiza la Fase de Control, se solicita al Agente modificar una o varias variables MIB, por ejemplo la instancia correspondiente de la variable MIB ifAdminStatus (Estado Administrativo de la Interfaz). Esta variable sería la variable de control.

La modificación de la instancia de la variable de control correspondiente, se solicitaría al Agente solo cuando el resultado de la consulta indicara un valor diferente al valor actual de la instancia de la variable. Una instancia determinada de la variable ifAdminStatus, está asociada a un Interfaz de Red en particular, por ejemplo, listados en orden creciente: Loopback (.1), Lan (LLC NDIS) ó Wan (LLC PPP), si la computadora cuenta con Interfaz de Lan y Módem.

Una variable de control en esta investigación, debe satisfacer las siguientes características:

- Acceso Read Write. Debe tener la posibilidad de ser consultada y modificada.
- No Autobloqueo. Su modificación no debe causar un bloqueo en la operación del interfaz que evite su supervisión o control posterior.

La primera versión del script Administrador.tcl fué la siguiente:

```
# Administrador.tcl / wgo / 010710 / Version Inicial
# Script Tcl para Administracion Automatica de Agentes SNMP con Soporte Experto
##############################################################################
proc supervisarAgente {sesion} {
    ;# Instancia .1: LOOPBACK, next: LAN (LLC NDIS), next: WAN (LLC PPP)
    set s0 [$sesion getnext ifAdminStatus.1]       ;# Supervisa ifAdminStatus.LAN
    set s1 [$sesion getnext ifOperStatus.1]        ;# Supervisa ifOperStatus.LAN
    set s2 [$sesion getnext ifOutErrors.1]         ;# Supervisa ifOutErrors.LAN
    set s3 [$sesion getnext ifInErrors.1]          ;# Supervisa ifInErrors.LAN
    set s4 [$sesion getnext ifInUnknownProtos.1]   ;# Supervisa ifInUnknownProtos.LAN
    set s5 [$sesion get     sysContact.0]          ;# Supervisa sysContact.0

    set i0 [lindex [lindex $s1 0] 0]               ;# 1er elemento de 1ra lista (OID)
    set i1 [lindex [mib split $i0] 1]              ;# Instancia de OID (LAN)

    set v0 [lindex [lindex $s0 0] 2]               ;# Obtiene valor de Supervision 0
    set v1 [lindex [lindex $s1 0] 2]               ;# Obtiene valor de Supervision 1
    set v2 [lindex [lindex $s2 0] 2]               ;# Obtiene valor de Supervision 2
    set v3 [lindex [lindex $s3 0] 2]               ;# Obtiene valor de Supervision 3
    set v4 [lindex [lindex $s4 0] 2]               ;# Obtiene valor de Supervision 4
    set v5 [lindex [lindex $s5 0] 2]               ;# Obtiene valor de Supervision 5
    set listaDeRespuestas [list $v0 $v1 $v2 $v3 $v4 $v5 $i1]
    return $listaDeRespuestas                      ;# Regresa Lista de Respuestas
}
##############################################################################
package require Tnm 3.0                           ;# Carga paquete Tnm 3.0 en Tcl/Tk
namespace import Tnm::*                           ;# Permite snmp por Tnm::snmp, ...

set numCiclos 2                                   ;# Ciclos de Trabajo
set listaDeAgentes {127.0.0.1 192.168.0.1}        ;# Lista de Direcciones IP de Agentes

for {set c 0} {$c < $numCiclos} {incr c} {          ;# Ejecuta un numero de Ciclos
   foreach agente $listaDeAgentes {                 ;# Por cada agente en la Lista
      puts -nonewline "Ciclo: $c, Agente: $agente, "
      set sesion [snmp generator -address $agente]  ;# Abre una Sesion con un Agente
      puts "Sesion: $sesion"

      ;################################################ Fase de Supervision
      set listaValores [supervisarAgente $sesion]
      puts "\t Resultado de la Supervision:"; puts "\t $listaValores"
                                                    ;# Valor de:
      set v0 [lindex $listaValores 0]               ;#   ifAdminStatus.Instancia
      set v1 [lindex $listaValores 1]               ;#   ifOperStatus.Instancia
      set v2 [lindex $listaValores 2]               ;#   ifOutErrors.Instancia
      set v3 [lindex $listaValores 3]               ;#   ifInErrors.Instancia
      set v4 [lindex $listaValores 4]               ;#   ifInUnknownProtos.Instancia
      set v5 [lindex $listaValores 5]               ;#   sysContact.0
      set i1 [lindex $listaValores 6]               ;#   Instancia

      ;################################################ Fase de Consulta
      set consulta ""
      set consulta [exec InterfazAut ExpertoAut $v1 $v2 $v3 $v4 $v5]
      puts "\t Resultado de la Consulta: $consulta"

      ;################################################ Fase de Control
      if {$consulta != $v0} {
         set varControl [list [list ifAdminStatus.$i1 "OCTET STRING" $consulta]]
```

```
        set control [$sesion set $listaVar]
      } else {
        set control "No necesaria"
      }
      puts "\t Accion de Control: $control"

      ;############################################## Destruye la Sesion
      $sesion destroy                          ;# Destruye la Sesion
      after 5000                               ;# Espera 5,000 ms
    }                                          ;# Siguiente Agente
}                                              ;# Siguiente Ciclo
```

Este script se ejemplificó con dos ciclos para una cierta Lista de Agentes, usando como variable de control a las instancias correspondientes de ifAdminStatus.

Al ejecutarse el script Administrador.tcl se obtuvieron resultados mixtos. Las partes de supervisión y consulta operaban correctamente, pero cuando se modificaba el valor de la instancia Lan de la variable de control con el valor "no", causaba un bloqueo en la operación del Interfaz Lan que impedía su posterior supervisión. Se concluyó que la variable ifAdminStatus no era una variable apropiada para operar la parte de control.

Una variable de control que permitiera mostrar la tecnología de interés y que se usara principalmente cuando la Base de Conocimientos se empezara a desarrollar, podría ser una variable que controlara el reinicio (reboot) del Interfaz, operación que cuando se incrementara la Base de Conocimientos solo se realizaría en casos extremos. Esto implicaría, la necesidad de generar una MIB en particular y que los objetos administrables soportaran la operación de reinicio a través de una variable MIB.

Dado que esta investigación tiene como objetivo principal mostrar la posibilidad de soporte experto automático en la administración de redes, se buscó una variable que permitiera:

- Mostrar la tecnología de control sobre el Agente SNMP,
- No bloquear a ningún dispositivo,
- No requirir una MIB particular y
- Ser fácilmente observable.

Las variables que se detectaron con tales posibilidades fueron sysContact y sysLocation. Ambas variables tienen una instancia única por Agente, la instancia 0. Como la instancia sysContact.0 ya se usa en la Base de Conocimientos, se decidió el uso de sysLocation.0 como la instancia de la variable de control para efectos de prueba. Considérese que para efectos de prueba sysLocation.0 manejará los valores "up" ó "no" que manejaba anteriormente ifAdminStatus. La versión final del script Administrador.tcl es la siguiente:

```
# Administrador.tcl / wgo / 010710 / Version Final para Prueba*
# Script Tcl para Administracion Automatica de Agentes SNMP con Soporte Experto
##############################################################################
proc supervisarAgente {sesion} {
    ;# Instancia .1: LOOPBACK, next: LAN (LLC NDIS), next: WAN (LLC PPP)

    ;#* set s0 [$sesion getnext ifAdminStatus.1]   ;# Supervisa ifAdminStatus.LAN
    set s0 [$sesion get     sysLocation.0]         ;#* Supervisa sysLocation.0

    set s1 [$sesion getnext ifOperStatus.1]        ;# Supervisa ifOperStatus.LAN
    set s2 [$sesion getnext ifOutErrors.1]         ;# Supervisa ifOutErrors.LAN
    set s3 [$sesion getnext ifInErrors.1]          ;# Supervisa ifInErrors.LAN
```

```
    set s4 [$sesion getnext ifInUnknownProtos.1]   ;# Supervisa ifInUnknownProtos.LAN
    set s5 [$sesion get     sysContact.0]          ;# Supervisa sysContact.0
    set i0 [lindex [lindex $s1 0] 0]               ;# 1er elemento de 1ra lista (OID)
    set i1 [lindex [mib split $i0] 1]              ;# Instancia de OID (LAN)
    set v0 [lindex [lindex $s0 0] 2]               ;# Obtiene valor de Supervision 0
    set v1 [lindex [lindex $s1 0] 2]               ;# Obtiene valor de Supervision 1
    set v2 [lindex [lindex $s2 0] 2]               ;# Obtiene valor de Supervision 2
    set v3 [lindex [lindex $s3 0] 2]               ;# Obtiene valor de Supervision 3
    set v4 [lindex [lindex $s4 0] 2]               ;# Obtiene valor de Supervision 4
    set v5 [lindex [lindex $s5 0] 2]               ;# Obtiene valor de Supervision 5
    set listaDeRespuestas [list $v0 $v1 $v2 $v3 $v4 $v5 $i1]
    return $listaDeRespuestas                      ;# Regresa Lista de Respuestas
}
##############################################################################
package require Tnm 3.0                          ;# Carga paquete Tnm 3.0 en Tcl/Tk
namespace import Tnm::*                          ;# Permite snmp por Tnm::snmp, ...

set numCiclos 2                                  ;# Ciclos de Trabajo
set listaDeAgentes {192.168.0.1 192.168.0.1}     ;#* Lista de Direcciones IP de Agentes

for {set c 0} {$c < $numCiclos} {incr c} {          ;# Ejecuta un numero de Ciclos
   foreach agente $listaDeAgentes {                 ;# Por cada agente en la Lista
      puts -nonewline "Ciclo: $c, Agente: $agente, "
      set sesion [snmp generator -address $agente]  ;# Abre una Sesion con un Agente
      puts "Sesion: $sesion"
      ;################################################ Fase de Supervision
      set listaValores [supervisarAgente $sesion]
      puts "\t Resultado de la Supervision:"; puts "\t $listaValores"
                                                    ;# Valor de:
      set v0 [lindex $listaValores 0]               ;#*  sysLocation.0
      set v1 [lindex $listaValores 1]               ;#   ifOperStatus.Instancia
      set v2 [lindex $listaValores 2]               ;#   ifOutErrors.Instancia
      set v3 [lindex $listaValores 3]               ;#   ifInErrors.Instancia
      set v4 [lindex $listaValores 4]               ;#   ifInUnknownProtos.Instancia
      set v5 [lindex $listaValores 5]               ;#   sysContact.0
      set i1 [lindex $listaValores 6]               ;#   Instancia
      ;#* set v0 [lindex $listaValores 6]           ;#   ifAdminStatus.Instancia
      ;################################################ Fase de Consulta
      set consulta ""
      set consulta [exec InterfazAut ExpertoAut $v1 $v2 $v3 $v4 $v5]
      puts "\t Resultado de la Consulta: $consulta"
      ;################################################ Fase de Control
      if {$consulta != $v0} {
         ;#* set varControl [list [list ifAdminStatus.$i1 "OCTET STRING" $consulta]]
         set varControl [list [list sysLocation.0 "OCTET STRING" $consulta]]    ;#*

         set control [$sesion set $varControl]
      } else {
         set control "No necesaria"
      }
      puts "\t Accion de Control: $control"
      ;################################################ Destruye la Sesion
      $sesion destroy                           ;# Destruye la Sesion
      after 5000                                ;# Espera 5,000 ms
   }                                            ;# Siguiente Agente
}                                               ;# Siguiente Ciclo
```

6.3. El Sistema Experto.

La consulta al Sistema Experto la realiza el Sistema Administrador por medio de la ejecución del programa InterfazAuto.exe.

Cuando se ejecuta este programa se indica *InterfazAut ExpertoAut $v1 $v2 $v3 $v4 $v5*, lo cual implica la recepción de 6 parámetros:

- *ExpertoAut* Este es el Programa Experto Enlazado. Este programa se obtiene a partir de un programa fuente en Prolog (ExpertoAut.pro) que se compila (ExpertoAut.plm) y se enlaza (ExpertoAut.xpl) bajo la versión Prolog Amzi 5.0.
- *$v1 $v2 ... $v5* Esta es la parte dinámica de la Base de Hechos. Se introducen 5 hechos en tiempo de ejecución: ifOperStatus, ifOutErrors, ifInErrors, ifInUnknownProtos y sysContact. Las instancias de las primeras corresponden a un solo Interfaz. La instancia de sysContact corresponde al Agente y es única.

El programa experto debe permitir la introducción de hechos de manera dinámica dado que dependerán de cada consulta, por lo que se debe preparar la posibilidad de introducir o eliminar un término de esta manera, además de permitir la discontinuidad en los términos si la versión de Prolog así lo requiere, a fin de permitir la entrada de hechos en cualquier secuencia. Estas dos operaciones se realizan por medio de las instrucciones *dynamic termino/aridad* y *discontiguos termino/aridad* en la versión Prolog Amzi 5.0.

Los hechos se introducirán en tiempo de ejecución a través del Interfaz de Operación por medio de la instrucción *asserta(termino(argumento))*. Los 5 hechos (*termino/aridad*) que se incluyen en cada consulta tienen aridad 1, es decir tienen un solo argumento.

El programa experto fuente (ExpertoAut.pro) fue modificado para considerar el cambio de variable de control de ifAdminStatus a sysLocation. La versión final es la siguiente:

```
% ExpertoAut.pro / wgo / 010710 / Version Final para Prueba*
% Esquema de representacion de datos: atributo(valor)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Base de Hechos (Parte Dinamica)
dynamic ifOperStatus/1.
dynamic ifOutErrors/1.
dynamic ifInErrors/1.
dynamic ifInUnknownProtos/1.
dynamic sysContact/1.
discontiguous ifOperStatus/1.
discontiguous ifOutErrors/1.
discontiguous ifInErrors/1.
discontiguous ifInUnknownProtos/1.
discontiguous sysContact/1.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Base de Conocimientos %%%%%%%%%
sysLocation(up)    :-  ifOperStatus(up),
                       ifOutErrors(V0), maxOutErrors(V0M), V0 =< V0M,
                       ifInErrors(V1), maxInErrors(V1M), V1 =< V1M,
                       ifInUnknownProtos(V2), maxInUnknownProtos(V2M), V2 =< V2M,
                       sysContact(V3), authContact(V3).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Base de Hechos (Parte Estatica)
maxOutErrors(1).
maxInErrors(5).
maxInUnknownProtos(200).
authContact(beatrizTrejoMonroy).
authContact(wenceslaoGrandeOlguin).
```

El programa fuente ExpertoAut.pro se compila (ExpertoAut.plm) y posteriormente se enlaza (ExpertoAut.xpl) desde la opción *Build* de la Barra de Menú del Entorno Interactivo de Desarrollo (Interactive Development Environment) de Amzi! Prolog+Logic Server versión 5.0. Una vez preparado el programa ExpertoAut.xpl, su ejecución se realiza de manera automática, a través del Interfaz de Operación del Sistema Experto.

6.4. El Interfaz de Operación.

El Interfaz de Operación del Sistema Experto permite consultar al programa experto incrustado (ExpertoAut.xpl) de manera automática. Para interaccionar con el programa incrustado desarrollado en Prolog (compilado y enlazado) a través del Interfaz de Operación desarrollado en C, se debe incluir el archivo de encabezado *amzi.h* y usar las funciones predefinidas *ls*()* del Servidor Lógico de Amzi! Prolog+Logic Server.

El Servidor Lógico es el mecanismo que permite la comunicación entre el programa anfitrión en C y el programa incrustado en Prolog.

El programa en Lenguaje C del Interfaz de Operación inicia incluyendo archivos de encabezado para operaciones de entrada y salida (stdio.h), operaciones cadena (string.h) y operaciones con el Servidor Lógico (amzi.h). Enseguida se introduce la función principal (main) y se declaran las variables requeridas para la operación del Servidor Lógico y del programa en general.

```
// InterfazAut.c / wgo / 010710 / Version Final para Prueba*
// Ejecucion:
// InterfazAut ExpertoAut ifOperStatus ifOutErrors ifInErrors ifInUnknownProtos sysContact
// Ejemplo: InterfazAut ExpertoAut up 0 1 100 wenceslaoGrandeOlguin

#include <stdio.h>
#include <string.h>
#include "amzi.h"

int main(int numArg, char *valArg[])

        ENGid   motorDeInferencia;
        TERM    terminoLogico;
        TF      resultadoLogico;
        int     resultadoEntero;

        char    Experto[50]                 = "";
        char    ifOperStatus[50]            = "";
        char    ifOutErrors[50]             = "";
        char    ifInErrors[50]              = "";
        char    ifInUnknownProtos[50]       = "";
        char    sysContact[50]              = "";
        char    consulta[50]                = "";
        char    resultadoConsulta[50]       = "";
```

Posteriormente se inicializan los servicios del Servidor Lógico y se carga el programa enlazado *ExpertoAut.xpl* (almacenado en valArg[1]) localizado en el mismo directorio que InterfazAut.exe. El programa ExpertoAut se asocia con un identificador denominado *motorDeInferencia.*

```
// Inicializacion del Servidor Logico
resultadoEntero = lsInit(&motorDeInferencia, "");
// if (resultadoEntero == 0)
//                   printf("Inicializando el Servidor Logico ...\n");

// Carga del Programa ExpertoAut *.xpl (Compilado y Enlazado)
// Localizado en el mismo Directorio que InterfazAut.exe
resultadoEntero = lsLoad(motorDeInferencia, valArg[1]);
// if (resultadoEntero == 0)
//                   printf("Cargando el Programa Experto: %s ...\n", valArg[1]);
```

Observe que se han comentado algunas instrucciones del programa que se usaron para supervisar su ejecución.

En esta parte del programa se capturan los hechos que se recibieron como parámetros.

```
// Captura del hecho: ifOperStatus(valArg[2])
strcat(ifOperStatus, "ifOperStatus(");
strcat(ifOperStatus, valArg[2]);
strcat(ifOperStatus, ")");

// Captura del hecho: ifOoutErrors(valArg[3])
strcat(ifOutErrors, "ifOutErrors(");
strcat(ifOutErrors, valArg[3]);
strcat(ifOutErrors, ")");

// Captura del hecho: ifInErrors(valArg[4])
strcat(ifInErrors, "ifInErrors(");
strcat(ifInErrors, valArg[4]);
strcat(ifInErrors, ")");

// Captura del hecho: ifInUnknownProtos(valArg[5])
strcat(ifInUnknownProtos, "ifInUnknownProtos(");
strcat(ifInUnknownProtos, valArg[5]);
strcat(ifInUnknownProtos, ")");

// Captura del hecho: sysContact(valArg[6])
strcat(sysContact, "sysContact(");
strcat(sysContact, valArg[6]);
strcat(sysContact, ")");
```

Los hechos se incluyen en la Base de Hechos con el código siguiente.

```
// Inclusion de Hechos
resultadoEntero = (lsAssertaStr(motorDeInferencia, ifOperStatus));
resultadoEntero = (lsAssertaStr(motorDeInferencia, ifOutErrors));
resultadoEntero = (lsAssertaStr(motorDeInferencia, ifInErrors));
resultadoEntero = (lsAssertaStr(motorDeInferencia, ifInUnknownProtos));
resultadoEntero = (lsAssertaStr(motorDeInferencia, sysContact));

// printf("%s\n", ifOperStatus);
// printf("%s\n", ifOutErrors);
// printf("%s\n", ifInErrors);
// printf("%s\n", ifInUnknownProtos);
// printf("%s\n", sysContact);
```

La consulta del valor de sysLocation se maneja con la codificación siguiente:

```
// Consulta cual debe ser el valor de sysLocation
strcat(consulta, "sysLocation(X)");
resultadoLogico = lsCallStr(motorDeInferencia, &terminoLogico, consulta);
// printf("%s\n", consulta);
```

Dependiendo del número de soluciones de interés, la respuesta puede ser de dos tipos. Un tipo de respuesta permite obtener todas las soluciones posibles, mientras que el otro tipo de respuesta solo obtiene la primer solución. Ambas codificaciones se muestran enseguida.

En cada ocasión que exista unificación, la variable *resultadoLogico* contendrá el valor *TRUE*.

Cuando se requiren todas las soluciones posibles se puede usar el siguiente código:

```
// Obtener todas las Soluciones Posibles
while (resultadoLogico == TRUE) {
resultadoEntero =
lsGetArg(motorDeInferencia, terminoLogico, 1, cSTR, resultadoConsulta);
        // (motorDeInferencia, terminoLogico, aridad, tipoDeDato, resultadoConsulta);
        printf("%s\n", resultadoConsulta);
        // Alguna otra respuesta ?
        resultadoLogico = lsRedo(motorDeInferencia);
};
strcpy(resultadoConsulta, "no");
printf("%s\n", resultadoConsulta);
```

Si solo se requiere la primer solución, como en la presente investigación, se usa este código:

```
// Obtener solo la Primer Respuesta
if (resultadoLogico == TRUE) {
        resultadoEntero =
        lsGetArg(motorDeInferencia, terminoLogico, 1, cSTR, resultadoConsulta);
        // (motorDeInferencia, terminoLogico, aridad, tipoDeDato, resultadoConsulta);
} else {
        strcpy(resultadoConsulta, "no");
};
printf("%s\n", resultadoConsulta);
```

Finalmente se cierra el Servidor Lógico y se termina el programa.

```
        // Finalizacion del Servidor Logico
        lsClose(&motorDeInferencia);

        // Fin del Programa
        return(0);
}
```

El programa del Interfaz de Operación del Sistema Experto se desarrolló bajo Microsoft Visual C++ 6.0 [41][42]. El archivo de encabezado "amzi.h" se puede obtener de la página http://www.amzi.com. La versión completa del Interfaz de Operación del Sistema Experto se muestra enseguida.

```
// InterfazAut.c / wgo / 010710 / Version Final para Prueba*
// Ejecucion:
// InterfazAut ExpertoAut ifOperStatus ifOutErrors ifInErrors ifInUnknownProtos sysContact
// Ejemplo: InterfazAut ExpertoAut up 0 1 100 wenceslaoGrandeOlguin

#include <stdio.h>
#include <string.h>
#include "amzi.h"

int main(int numArg, char *valArg[]) {

        ENGid   motorDeInferencia;
        TERM    terminoLogico;
        TF      resultadoLogico;
        int     resultadoEntero;
```

```
char    Experto[50]                 = "";
char    ifOperStatus[50]            = "";
char    ifOutErrors[50]             = "";
char    ifInErrors[50]              = "";
char    ifInUnknownProtos[50]       = "";
char    sysContact[50]              = "";
char    consulta[50]                = "";
char    resultadoConsulta[50]       = "";

// Inicializacion del Servidor Logico
resultadoEntero = lsInit(&motorDeInferencia, "");
// if (resultadoEntero == 0)
//                   printf("Inicializando el Servidor Logico ...\n");

// Carga del Programa ExpertoAut *.xpl (Compilado y Enlazado)
// Localizado en el mismo Directorio que InterfazAut.exe
resultadoEntero = lsLoad(motorDeInferencia, valArg[1]);
// if (resultadoEntero == 0)
//                   printf("Cargando el Programa Experto: %s ...\n", valArg[1]);

// Captura del hecho: ifOperStatus(valArg[2])
strcat(ifOperStatus, "ifOperStatus(");
strcat(ifOperStatus, valArg[2]);
strcat(ifOperStatus, ")");

// Captura del hecho: ifOoutErrors(valArg[3])
strcat(ifOutErrors, "ifOutErrors(");
strcat(ifOutErrors, valArg[3]);
strcat(ifOutErrors, ")");

// Captura del hecho: ifInErrors(valArg[4])
strcat(ifInErrors, "ifInErrors(");
strcat(ifInErrors, valArg[4]);
strcat(ifInErrors, ")");

// Captura del hecho: ifInUnknownProtos(valArg[5])
strcat(ifInUnknownProtos, "ifInUnknownProtos(");
strcat(ifInUnknownProtos, valArg[5]);
strcat(ifInUnknownProtos, ")");

// Captura del hecho: sysContact(valArg[6])
strcat(sysContact, "sysContact(");
strcat(sysContact, valArg[6]);
strcat(sysContact, ")");

// Inclusion de Hechos
resultadoEntero = (lsAssertaStr(motorDeInferencia, ifOperStatus));
resultadoEntero = (lsAssertaStr(motorDeInferencia, ifOutErrors));
resultadoEntero = (lsAssertaStr(motorDeInferencia, ifInErrors));
resultadoEntero = (lsAssertaStr(motorDeInferencia, ifInUnknownProtos));
resultadoEntero = (lsAssertaStr(motorDeInferencia, sysContact));

// printf("%s\n", ifOperStatus);
// printf("%s\n", ifOutErrors);
// printf("%s\n", ifInErrors);
// printf("%s\n", ifInUnknownProtos);
// printf("%s\n", sysContact);

// Consulta cual debe ser el valor de sysLocation
strcat(consulta, "sysLocation(X)");
resultadoLogico = lsCallStr(motorDeInferencia, &terminoLogico, consulta);
// printf("%s\n", consulta);

// Obtener solo la Primer Respuesta
if (resultadoLogico == TRUE) {
       resultadoEntero =
       lsGetArg(motorDeInferencia, terminoLogico, 1, cSTR, resultadoConsulta);
       // (motorDeInferencia, terminoLogico, aridad, tipoDeDato, resultadoConsulta);
} else {
       strcpy(resultadoConsulta, "no");
```

```
        };
        printf("%s\n", resultadoConsulta);

        // Finalizacion del Servidor Logico
        lsClose(&motorDeInferencia);

        // Fin del Programa
        return(0);
}
```

6.5. La Prueba del Modelo.

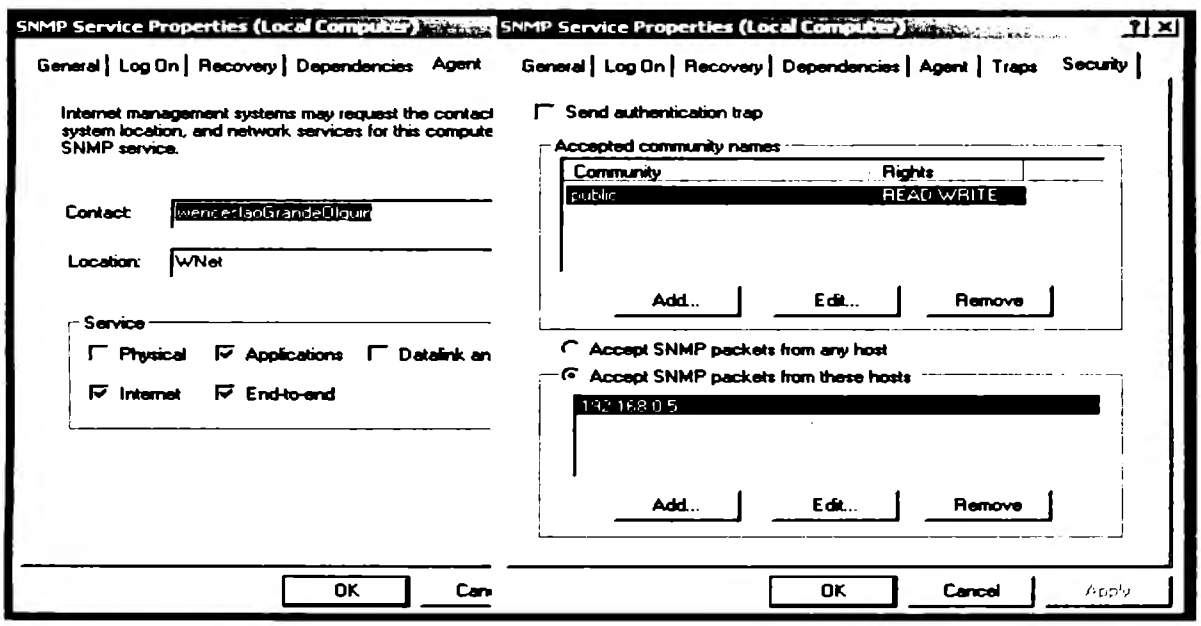
La evidencia de la operación del Modelo Agente-Administrador-Experto se desarrolló sobre una red TCP/IP donde el Agente SNMP opera sobre una computadora personal con Windows 2000 Advanced Server y el Sistema de Administración junto con el Sistema Experto operan sobre una computadora personal con Windows 2000 Me.

El Agente SNMP funciona con el Modelo Agente-Administrador-Experto independientemente de su plataforma de operación pero, al usar Windows 2000 Advanced Server, los Servicios SNMP se pueden configurar y consultar muy fácilmente.

La configuración y visualización de los Servicios SNMP en el Agente Windows 2000 Advanced Server, se realiza en

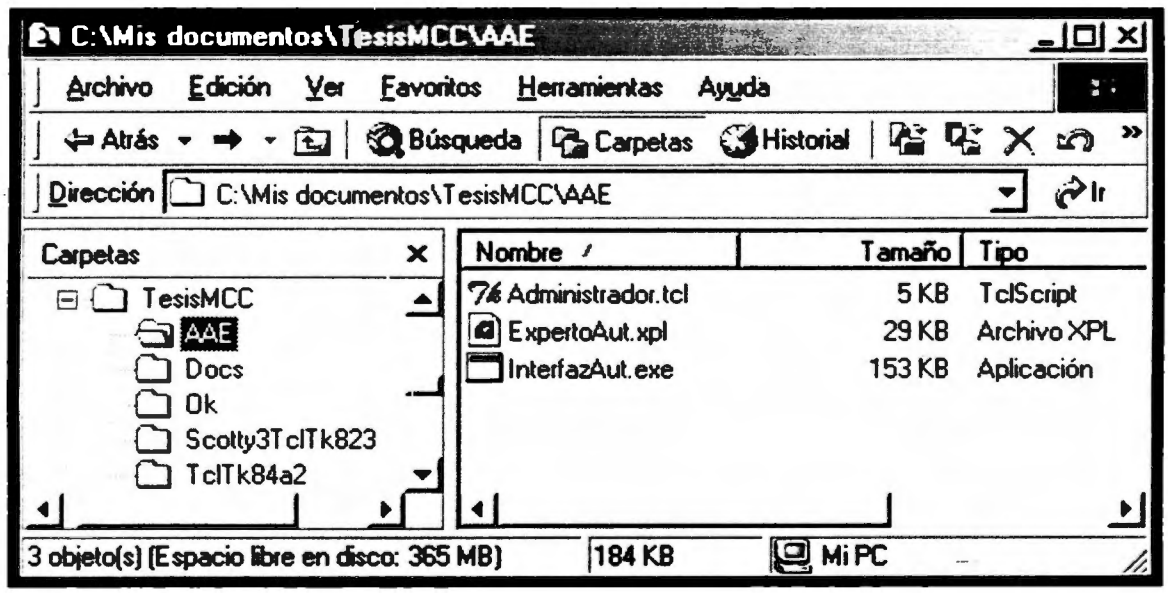
Control Panel | *Administrative Tools* | *Component Services* | *Services (Local)* | *SNMP Service*

usando principalmente dos ventanas, la ventana *Agent* y la ventana *Security*. En la ventana *Agent* se define el nombre del contacto (sysContact) y la localidad (sysLocation). En la ventana *Security* se indica el nombre de la comunidad con sus derechos asociados y opcionalmente las identificaciones de los nodos desde donde se pueden aceptar mensajes SNMP según se muestra en la siguiente figura.



El Sistema de Administración requiere que se instale el software Tcl/Tk y el software para administración de redes Scotty, para poder realizar las fases de supervisión y control. El script Administrador.tcl es el programa administrador que se usará en la prueba.

Para la fase de consulta, se requiere que el Sistema Experto cuente con el programa ejecutable del Interfaz de Operación (*InterfazAut.exe)* y el programa enlazado del Sistema Experto *(ExpertoAut.xpl)*. Estos programas junto con el script *Administrador.tcl*, se colocaron en un mismo directorio para su ejecución.



La prueba del Modelo Agente-Administrador-Experto se ejecutó desde una ventana MsDos invocando al intérprete textual Tclsh82.exe y proporcionándole como argumento el script Administrador.tcl.

En la Fase de Supervisión se muestran los valores actuales de las seis instancias de las variables MIB de interés del Agente y el valor de la instancia LAN:

```
sysLocation.0,
ifOperStatus.instanciaLAN,
ifOutErrors. instanciaLAN,
ifInErrors. instanciaLAN,
ifInUnknownProtos. instanciaLAN,
sysContact.0 e
instanciaLAN.
```

En la Fase de Consulta se muestra el valor de la respuesta. Un resultado *up* significa que estan dadas las condiciones para que la Interfaz se active, un resultado *no* significa que no están dadas tales condiciones.

En la Fase de Control se muestra la acción de control a realizar sobre la variable de control sysLocation. El criterio para ordenar alguna acción es que el resultado de la consulta sea diferente al valor de sysLocation.0. Si el resultado de la consulta es igual al valor actual de sysLocation.0 no es necesario ordenar ninguna acción de control. Las acciones de control solicitan modificar el valor de sysLocation.0, para almacenar el valor "up" o el valor "no".

set {1.3.6.1.2.1.1.6.0 {OCTET STRING} up},
set {1.3.6.1.2.1.1.6.0 {OCTET STRING} no} o
No necesaria

El siguiente desplegado de MsDos muestra cuando el resultado de la consulta es "no" y el valor de sysLocation.0 es "WNet" lo cual lleva a una acción de control para modificar el valor de sysLocation.0 a "no". También muestra cuando el resultado de la consulta es "no" y el valor de sysLocation.0 es "no" lo que hace innecesaria una acción de control. Recuérdese que el primer resultado de la supervisión es el valor de sysLocation.0.

```
C:\Mis documentos\TesisMCC\AAE> tclsh82 Administrador.tcl
Ciclo: 0, Agente: 192.168.0.1, Sesion: snmp0
       Resultado de la Supervision:
       WNet 0 0 0 otroNombre 16777219
       Resultado de la Consulta: no
       Accion de Control: {1.3.6.1.2.1.1.6.0 {OCTET STRING} no}
Ciclo: 0, Agente: 192.168.0.1, Sesion: snmp1
       Resultado de la Supervision:
       no up 0 0 0 otroNombre 16777219
       Resultado de la Consulta: no
       Accion de Control: No necesaria
Ciclo: 1, Agente: 192.168.0.1, Sesion: snmp2
       Resultado de la Supervision:
       no up 0 0 0 otroNombre 16777219
       Resultado de la Consulta: no
       Accion de Control: No necesaria
Ciclo: 1, Agente: 192.168.0.1, Sesion: snmp3
       Resultado de la Supervision:
       no up 0 0 0 otroNombre 16777219
       Resultado de la Consulta: no
       Accion de Control: No necesaria
```

El desplegado siguiente muestra cuando el resultado de la consulta es "up" y el valor de sysLocation.0 es "WNet" lo cual lleva a una acción de control que modifica el valor de sysLocation.0 a "up". También se muestra cuando el resultado de la consulta es "up" y el valor de sysLocation también es "up" lo que hace innecesaria una acción de control.

```
C:\Mis documentos\TesisMCC\AAE> tclsh82 Administrador.tcl
Ciclo: 0, Agente: 192.168.0.1, Sesion: snmp0
       Resultado de la Supervision:
       WNet up 0 0 0 wenceslaoGrandeOlguin 16777219
       Resultado de la Consulta: up
       Accion de Control: {1.3.6.1.2.1.1.6.0 {OCTET STRING} up}
Ciclo: 0, Agente: 192.168.0.1, Sesion: snmp1
       Resultado de la Supervision:
       up up 0 0 0 wenceslaoGrandeOlguin 16777219
       Resultado de la Consulta: up
```

```
       Accion de Control: No necesaria
Ciclo: 1, Agente: 192.168.0.1, Sesion: snmp2
       Resultado de la Supervision:
       up up 0 0 0 wenceslaoGrandeOlguin 16777219
       Resultado de la Consulta: up
       Accion de Control: No necesaria
Ciclo: 1, Agente: 192.168.0.1, Sesion: snmp3
       Resultado de la Supervision:
       up up 0 0 0 wenceslaoGrandeOlguin 16777219
       Resultado de la Consulta: up
       Accion de Control: No necesaria
```

Estos desplegados muestran la viabilidad de proporcionar soporte experto a un sistema administrador de redes de manera automática, lográndose de esta manera el objetivo principal de la presente investigación.

7. Conclusiones.

El objetivo de la presente investigación, mostrar la viabilidad de proporcionar soporte experto a un sistema de administración de redes de manera automática, se cumple y es factible de implementarse en sistemas operados por usuarios finales. Los resultados y especialmente el desarrollo de la investigación, han permitido introducirnos en un área de alto interés en el campo de la administración de redes de computadoras.

Tomando como base los conocimientos proporcionados en las materias que integran el Plan de Estudios de la Maestría en Ciencias de la Computación, se realizó la investigación usando múltiples fuentes de información, considerando de manera especial a los RFC's relacionados con el Protocolo SNMP, el cual sirve de infraestructura para realizar las tareas de la administración de redes.

Para poder comprender la documentación sobre el Protocolo SNMP, fue necesario analizar a las normas ASN.1, usadas para definir la sintaxis abstracta de los objetos administrables y la sintaxis de transferencia de los mensajes.

Una vez sentadas las bases, se estudio la filosofía de operación del Protocolo SNMP. SNMP opera por medio de mensajes que permiten supervisar y controlar variables normalizadas en los objetos administrables. Se analizó también, como tal filosofía se plasma en una infraestructura simple pero eficaz en la versión SNMPv1, la cual se ha convertido en la más popular, particularmente para realizar operaciones de supervisión.

La búsqueda de un mecanismo para lograr que diferentes aplicaciones se integraran fácilmente en un sistema, llevó a considerar diversas alternativas que culminaron con la decisión de usar al Lenguaje Tcl como elemento integrador.

Los lenguajes script como Tcl, usan programas existentes en áreas donde sus características se explotan a plenitud y los integran para multiplicar sus beneficios. Las facilidades de Tcl para manejar comunicaciones bajo el conjunto de Protocolos TCP/IP, fué una característica de suma utilidad para lograr el objetivo de la investigación. Como un resultado lateral de la investigación, se desarrolló un sistema que opera el puerto paralelo de una computadora remota, usando las facilidades de comunicación por sockets de Tcl.

Debe comentarse que la extensión Tk de Tcl, la cual permite un interfaz gráfica de usuario, es una facilidad que se está explotando e inclusive importando por otros lenguajes (Perl/Tk) y que permite, de una manera relativamente sencilla, manejar un entorno visual en ambientes Windows y Unix.

Para administrar la red se requería de una implementación del Protocolo SNMP que operara en las plataformas operativas más comunes en la actualidad, con facilidades profesionales y con un costo accesible para la mayoría de organizaciones de nuestro entorno que contaran con redes. Esto llevó a considerar las recomendaciones de los expertos y probar Scotty. Scotty es un producto gratuito que opera en Windows y Unix de manera eficaz, sin embargo, la calidad de su documentación es limitada. Se operó bajo prueba y error hasta alcanzar una comprensión

razonable del sistema. Se hicieron extensas pruebas de operación, principalmente de la parte textual, hasta ratificar la comprensión del sistema.

Los Sistemas Expertos son un campo ya considerado dentro de las Ciencias de la Computación, pero es relativamente nueva su incursión en el campo de las redes de computadoras. Se logró investigar que existe la idea de desarrollar sistemas de administración de redes con soporte experto, pero no se logró obtener ningún desarrollo.

Aún cuando el Sistema Experto mostrado en la investigación es mínimo, presenta la idea básica de codificar en premisas conocimiento experto sobre redes y obtener conclusiones que permitan ejercer control sobre los elementos administrables de una red.

El sistema experto se probó de manera interactiva, pero para operarse de manera automática requirió de una Interfaz de Operación en Lenguaje C. Esta Interfaz de Operación permite consultar automáticamente al experto desarrollado en Prolog a través de un mecanismo conocido como Servidor Lógico en la versión Amzi Prolog usada para tal efecto.

La integración de las diferentes aplicaciones se desarrollo bajo el Modelo Agente-Administrador-Experto. Este modelo conceptualiza la solución que ofrece la investigación. El modelo muestra a un Administrador operando en Ciclos de Supervision, Consulta y Control sobre una Lista de Agentes. La plataforma de prueba fué una Intranet con tecnologia TCP/IP. Windows fue el sistema operativo sobre el que corrió el Agente, el Administrador y el Experto.

Para desarrollar las pruebas se seleccionó inicialmente administrar un Interfaz de Red Local tipo Ethernet. El Interfaz de Red Local presentaba la dificultad de que si se bloqueaba administrativamente por medio de la variable de control respectiva, no podía ser desbloqueado a menos que se reinicializara. Se analizaron las alternativas y se decidió administrar la variable sysLocation.0, la cual si permitía mostrar la tecnología, no bloqueaba ningún dispositivo, era fácilmente observable y no tenía requerimientos adicionales.

Los resultados de la prueba muestran que si es factible proporcionar soporte experto a un sistema administrador de redes de manera automática.

8. Referencias.

8.1. Capítulo 1.

[1] Designing TCP/IP Internetworks, Geoff Bennett, Chapter 12, Van Nostrand Reinhold, 1995, TK5105.585.B46.

[2] SNMP, SNMPv2 and CMIP: The Practical Guide to Network-Management Standards, William Stallings, Addison-Wesley, 1993, TK5105.5.S763.

[3] TCP/IP and Related Protocols, Uyless Black, Chapter 11, McGraw-Hill, 1992, TK5105.5.B5663.

[4] SNMPv2, Simple or Sophisticated?, Vincent Berkhout, University of Twente, 1994.

[5] Expert Systems, John Durkin, McMillan, 1994, QA76.76.E95D87.

[6] C/C++ for Expert Systems, David Hu, Mis Press, 1989, QA.76.73.C15.H83.

8.2. Capítulo 2.

[7] Internet Engineering Task Force, www.ietf.org

[8] The Simple Times, Volumen 5, Number 1, December, 1997, www.simple-times.org

[9] A Practical Guide to SNMPv3 and Network Management, David Zeltserman, Prentice Hall PTR, TK5105.583.Z45 1999.

[10] ASN.1 Complete, John Larmouth, OS, 1999, www.asn1.com/asn1/larmouth.html

[11] ASN.1 site, asn1.elibel.tm.fr/en/introduction/index.htm

[12] The Open Book A Practical Perspective on Osi, Marshall T. Rose, Chapter 8, Prentice Hall, 1990, TK5105.5.R67.

[13] RFC 1155 - Structure of Management Information (SMI), www.ietf.org/rfc/rfc1155.txt

[14] RFC 1157 - Simple Network Management Protocol (SNMP), www.ietf.org/rfc/rfc1157.txt

[15] RFC 1213 - Management Information Base II (MIB-II), www.ietf.org/rfc/rfc1213.txt

[16] An Introductory Overview of SNMP, Diversified Data Resources, Inc., www.ddri.com/Doc/SNMP_Overview.html

[17] SNMP V1 Protocol Description, Atos, www.snmp-products.com/tutorials/snmpv1.html

[18] RFC 1212 - Concise MIB Definitions, www.ietf.org/rfc/rfc1212.txt

8.3. Capítulo 3.

[19] History of Tcl, John Ousterhout, www.scriptics.com/doc/tclHistory.html

[20] Sams Teach Yourself Tcl/Tk in 24 Hours, Venkat V.S.S. Sastry Ph.D. & Lakshmi Sastry Ph.D., Sams, 2000.

[21] Scripting: Higher Level Programming for the 21st Century, John K. Ousterhout. Ajuba Solutions, *IEEE Computer* magazine, March 1998, www.scriptics.com/doc/scripting.html

[22] How to manage your network using SNMP, Marshall T. Rose and Keith McCloghrie, Prentice Hall. 1995, TK5105.55.R67.

[23] Practical Programming in Tcl and Tk, Brent B. Welch, Prentice Hall PTR. QA76.73.T44 W45 1999.

[24] Tcl/Tk Pocket Reference, Paul Raines, O'Reilly, 1998.

[25] Tcl and the Tk Toolkit, John K. Ousterhout, Addison Wesley, QA76.73.T44097 1994.

[26] Tcl Developer Site, www.scriptics.com

[27] Tcl/Tk Community Site, tcltk.com/

[28] Avia Training and Consulting, Ken Jones, www.avia-training.com

[29] Tutorial de Tcl/Tk, Universidad de Oviedo, España, www.etsimo.uniovi.es/tcl/tutorial/

[30] Tcl Tutor: Interactive Computer Aided Instruction for Tcl, Clif Flynt, www.msen.com/~clif/TclTutor.html

8.4. Capítulo 4.

[31] SimpleWeb Site, www.simpleweb.org

[32] SNMPLink Site, www.snmplink.org

[33] Scotty, Juergen Schoenwaelder, University of Twente, wwwhome.cs.utwente.nl/~schoenw/scotty

[34] An Overview on the Scotty Management Package, Dr. Aiko Pras, University of Twente, June 2000.

[35] Building Network Management Tools with Tcl/Tk, Dave Zeltserman & Gerard Puoplo, Prentice Hall PTR, QA76.73.T44Z45 1998.

8.5. Capítulo 5.

[36] Building Expert Systems in Prolog, Dennis Merrit, www.amzi.com/ExpertSystemsInProlog

[37] Programming in Prolog, Clocksin and Mellish, Springer-Verlag.

[38] Prolog A Tutorial Introduction, James Lu & Jerry Mead, Bucknell University.

[39] SWI-Prolog, www.swi.psy.uva.nl/projects/SWI-Prolog

[40] Prolog Amzi, www.amzi.com

8.6. Capítulo 6.

[41] C++ Cómo Programar, Deitel y Deitel, Prentice Hall, 1999.

[42] Aprendiendo Visual C++ 5 en 21 Días, Gurewich & Gurewich, Prentice-Hall Hispanoamericana, 1998.