UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

SISTEMAS INTELIGENTES Y SU APLICACIÓN A LA GESTIÓN DE FALLAS EN REDES COMPLEJAS DE TELECOMUNICACIONES

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A:

LEONARDO SÁNCHEZ BOJÓRQUEZ

DIRECTOR DE TESIS:
DR. HUMBERTO ANDRÉS CARRILLO CALVET

2014

1. Datos del alumno
   Sánchez
   Bojórquez
   Leonardo
   55 95 62 01
   Universidad Nacional Autónoma de México
   Facultad de Ciencias
   Ciencias de la Computación
   099299327

2. Datos del tutor
   Dr
   Humberto Andrés
   Carrillo
   Calvet

3. Datos del sinodal 1
   Dr
   Juan Jaime
   Vega
   Castro

4. Datos del sinodal 2
   Dr
   José David
   Flores
   Peñaloza

5. Datos del sinodal 3
   Dr
   Felipe de Jesús
   Lara
   Rosano

6. Datos del sinodal 4
   M. en I.A.
   José Luis
   Jiménez
   Andrade

7. Datos del trabajo escrito
   Sistemas inteligentes y su aplicación a la gestión de fallas en redes complejas de telecomunicación
   (Sin subtítulo)
   108 p
   2014

Índice de contenido

Agradecimientos

Aunque a ciencia cierta desconocemos si la creatividad del universo es producto de la casualidad de las probabilidades o de la causalidad de una intención cuyo propósito estamos cumpliendo, en mi caso agradezco a Dios por experiencia sensacional que es la vida, la existencia y la herencia creativa que nos comparte en las ciencias y artes.

Gracias a todos los que han hecho posible el desarrollo de la tecnología trascendental que tanto me apasiona, porque nos acerca a la naturaleza misma del Universo. Esta tecnología que nos aleja de lo artificial de la mente humana para acercarnos al origen, nos aleja así mismo de la competencia, para llevarnos a la complementariedad de nuestra existencia con la realidad.

Quiero agradecer a mis padres y hermanos, por su amor, apoyo y educación, porque me han permitido vivir y alcanzar innumerables logros. Familia, ustedes son mi núcleo, LOS AMO A TODOS.

Gracias a Marce y Silver, que con su paciencia y conocimiento, lograron que saliera del bache en la realización de esta tesina, y me motivaron a enfrentar el gran honor de conocer al resto de mi familia, de modo que ahora puedo compartir este gran logro con ellos. ¡Los amo tíos, los amo primos!.

Silver y Rober, la entrega con la que impartieron sus clases de psicología y física, respectivamente, fueron el detonante perfecto para decidirme y enamorarme a estudiar las ciencias que rodean a la inteligencia artificial. ¡Muchas gracias!

Dan, *my brother from another mother!*, ¿qué habría sido de la carrera sin ti?. A todas las musas de mi vida, que definitivamente marcaron de manera positiva mi trayectoria en la carrera ¡Gracias por todo su apoyo!. Alber, eres la única persona con la que puedo hablar sobre el tema de la inteligencia artificial de manera plena. ¡Gracias por todos tus cuestionamientos y debrayes!

Gracias a todo el equipo del *Laboratorio de Dinámica No Lineal*, que sin su apoyo integral, no habría tenido esta gran oportunidad de titularme.

¡Gracias a todos mis amigos, que siempre han estado ahí para apoyarme!

Prefacio

Durante el seminario de titulación: "*Sistemas inteligentes y su aplicación a la administración de fallas en redes complejas de telecomunicación*", tuvimos como objetivo encontrar un área de oportunidad para el desarrollo científico de aplicaciones de software dentro de la industria de las telecomunicaciones. El Dr. Humberto Carrillo Calvet impartió este seminario durante el semestre 2012-1 y participamos las siguientes personas:

Profesores

Dr. Jaime Vega Castro.
M en I. A. José Luis Jiménez Andrade.
M. en C. Elio Villaseñor García.

Alumnos

Roberto García Medina.
John Henry Parker Gutiérrez.
Leonardo Sánchez Bojórquez.

Dentro del seminario, encontramos muy interesante el problema de la gestión automática de fallas dentro de las redes de telecomunicación. Dado que la mayoría de las fallas son frecuentes, pueden establecerse procedimientos que permitan corregirlas de manera automática. Esto reduce de manera considerable el tiempo de respuesta para su corrección y la cantidad de fallas que requieren de atención por parte de los técnicos que se encuentran monitorizando el estado de la red de telecomunicaciones desde un área de control centralizada. Aún así, la creciente demanda de servicios diversos y el crecimiento de la red han motivado a la industria a utilizar redes cada vez más grandes y complejas, lo cual requiere de nuevas tecnologías que permitan atender problemas más sofisticados.

Durante el seminario, entendimos que una gran cantidad de fallas son originadas por una falla raíz, y que al corregir a ésta falla, la corrección se propaga hacia otras fallas. Esto nos permite reducir de manera considerable el número de fallas que son consecuencia de una sola falla, pero el problema con estas fallas, es que no siempre se manifiestan de manera explícita. Es decir, tenemos que inferir la falla a través de la lectura de distintos síntomas en los elementos que conforman a la red de telecomunicaciones.

Existen distintas herramientas proporcionadas por tecnologías basadas en inteligencia artificial, que nos permiten automatizar la inferencia de fallas no explícitas, y así entonces poder encontrar las fallas raíz de distintos síntomas que se encuentran relacionados. Estas herramientas nos permiten generar nuevas fallas que relacionan a un grupo de fallas presentes. Este tipo de tecnología se aplica a distintas áreas como los son: telecomunicaciones, edificios inteligentes, industria automotriz, centrales de servicio de transporte, industria eléctrica, etc.

Dentro de este trabajo nos centraremos en el diseño de una arquitectura conformada por un contenedor

de mensajes de eventos de falla, un gestor de procesamiento de eventos en paralelo a través de unidades de procesamiento, que recolectarán información útil para los diagnósticos de fallas raíz no explícitas, y un sistema experto que mediante el uso de la información útil recolectada por las unidades de procesamiento, determinará cuando exista una falla raíz no explícita, para insertarla dentro del contenedor de mensajes de eventos de falla y darle tratamiento a través de las unidades de procesamiento.

Es importante aclarar, que estos sistemas de inteligencia artificial requieren del rol de un experto humano, que se encargue de codificar el conocimiento de los técnicos encargados de la administración de la red, para así integrar los procedimientos de identificación y corrección al sistema. Se puede pensar en utilizar técnicas de minería de datos para la identificación de causas raíz, con la finalidad de generar los procedimientos de identificación de causas, pero los resultados de estos análisis deberán estar sujetos al análisis del experto humano antes de integrarse al sistema en producción.

Se recomienda la lectura de los trabajos complementarios de mis dos compañeros, antes citados, para poder tener una imagen más completa acerca de los temas que revisamos en el seminario.

Esperamos que esta tesina pueda servir como un texto de apoyo a los seminarios del *Laboratorio de Dinámica no Lineal*, así como también para los cursos que sobre ésta temática sean impartidos en la *Facultad de Ciencias*.

Capítulo I

Introducción

Objetivos

La creciente demanda de servicios a través de redes de telecomunicaciones móviles, se ha visto reflejada en el incremento de la complejidad de sus operaciones administrativas. En el presente trabajo abordamos el problema de la corrección de fallas automatizada a través de métodos computaciones de inteligencia artificial, dentro de una red que se compone de elementos de distintos proveedores. Pues las soluciones actuales solo atienden a componentes de la red del mismo proveedor y adquirir las licencias para cada uno de los proveedores incrementa los costos de manera significativa.

Durante el seminario de titulación: *Sistemas inteligentes y su aplicación a la administración de fallas en redes complejas de telecomunicación*, estudiamos los distintos problemas que se presentan en las redes de telecomunicaciones. Como lo son la recepción masiva de mensajes de eventos de falla, su procesamiento en paralelo y generar una arquitectura que permita relacionar una serie de síntomas inconexos para realizar el diagnóstico de una falla cuya falla raíz no es evidente.

En el desarrollo del sistema que cubra las necesidades antes expuestas, no abordaremos las dificultades que existen para integrarlo dentro de un gestor de red, pues fundamentalmente sólo nos centraremos en el diseño de una arquitectura que nos permitirá satisfacer el problema descrito, ocupándonos de la gestión automática de mensajes de eventos a través de diversos métodos computacionales de inteligencia artificial para la corrección de fallas y correlación síntomas.

Es importante aclarar, que este sistema de inteligencia artificial requiere del rol de un experto humano, que se encargue de codificar el conocimiento de los técnicos encargados de la administración de la red, para así integrar los procedimientos de identificación y corrección al sistema. Se puede pensar en utilizar técnicas de minería de datos con la finalidad de generar los procedimientos de identificación de causas raíz, pero los resultados de estos análisis deben estar sujetos al análisis del experto humano antes de integrarse al sistema en producción.

En este trabajo, estudiaremos una arquitectura que resuelve el problema antes descrito, en donde la recepción de mensajes de eventos se realiza a través del desarrollo de un contenedor *FIFO* que permite su desborde a disco duro, el procesamiento en paralelo se logra mediante el desarrollo de un gestor de hilos para el procesamiento de los mensajes de eventos y la generación de diagnósticos para la correlación mediante una arquitectura con dos procesos simultáneos: el primer proceso atiende a los mensajes de eventos de manera local con máquinas de estados que recolectan información que podría ser útil para los diagnósticos complejos, y el segundo proceso analiza la información generada por el primer proceso con un sistema experto para determinar si existe alguna falla identificada en la correlación de síntomas.

La gestión de fallas dentro de las redes de telecomunicaciones

El objetivo de una red de telecomunicaciones es permitir la comunicación entre sus elementos, pero para que esto ocurra, se debe conformar una infraestructura que permita el envío del mensaje desde un punto de la red a otro. Esta infraestructura se compone de distintos elementos de red que permiten interconexiones temporales entre otros elementos que desean comunicarse entre sí, así como también se compone de elementos que permiten su monitorización, para que en caso de que ocurra un problema en el hardware o software se haga un mensaje con la notificación correspondiente.

Dentro de una red de telecomunicaciones, circulan una gran cantidad de mensajes que indican el estado operativo de la red misma, a estos mensajes se les llama *eventos*, algunos de estos eventos pueden ser *advertencias* con respecto al funcionamiento de la red, o bien pueden estar reportando la *fallas* de alguno de sus componentes.

Los eventos son capturados por un sistema de gestión de red denominado *OSS Operations Support Systems*, o Sistema de soporte a las operaciones, y cuando la red es grande, estos *OSS* pueden ser gestionados a su vez por un *MOM Manager of Managers*, o Manejador de manejadores. Ésta estructura jerárquica permite que los mensajes de eventos de falla puedan ser escalados dependiendo la complejidad relacionada a la corrección de la falla. Generalmente los *OSS* pueden filtrar mensajes de eventos irrelevantes y condensar mensajes redundantes, en algunos casos incluso pueden estar provistos por módulos que permiten el manejo de los mensajes de eventos de falla para su corrección, a estos módulos se les llama *FM Fault Managers*, o Manejadores de Fallas. Dado que no todos los *OSS* cuentan con un *FM*, se advierte la necesidad de un módulo para el manejo de fallas que pueda ser útil dentro de las distintas tecnologías que pueden conformar a una red de telecomunicaciones.

Gestión de fallas automática dentro de una red de telecomunicaciones

Dentro de la infraestructura de una red de telecomunicaciones, se cuenta con un centro de servicio que se encarga de mantener a la red operando en condiciones óptimas. Cuando la red es grande y compleja, este centro de servicio se apoya de las herramientas de monitorización que ofrecen los *OSS* para identificar problemas y solucionarlos.

Dado que los elementos de la red tienen una serie de fallas bien identificadas, es muy útil la creación de procedimientos específicos para la corrección de fallas frecuentes. Algunos de estos procedimientos pueden ser aplicados de manera automática por parte de sistemas expertos que se pueden encontrar dentro de los *OSS* en el *FM*. Manejar las fallas de manera automática ofrece muchos beneficios, de entre ellos está reducir el número de fallas que deberán ser atendidas por especialistas dentro del centro de servicio y la disminución del tiempo en la velocidad de respuesta.

Para el manejo de fallas, existe una técnica que se enfoca en encontrar la falla que es causa raíz dentro de una serie de fallas que se utilizan a manera de síntomas. Su efectividad reside en que al encontrar la causa raíz de muchas otras fallas y corregirla, el número total de fallas a atender se reduce considerablemente, incrementando la calidad en el diagnóstico y generación de estrategias para mantener en condiciones óptimas los servicios provistos por la red de telecomunicaciones.

Es importante mencionar que no todas las fallas que son causa raíz de otras fallas, son explícitas, es decir, el elemento de red no siempre realiza la generación del mensaje de falla que refiere a la falla raíz que el mismo origina. Esto hace necesaria la creación del mensaje de falla a partir de un diagnóstico por parte de un especialista o por parte de un sistema automático de inteligencia artificial dotado con el conocimiento necesario para identificar la falla.

En el desarrollo de un sistema que cubra las necesidades antes expuestas, no abordaremos las dificultades que existen para integrarlo dentro de un *OSS* pues fundamentalmente sólo nos centramos en el diseño de una arquitectura que nos permite satisfacer la recepción masiva de mensajes de eventos, su procesamiento en paralelo y la relación de síntomas inconexos para realizar el diagnóstico de una falla cuya falla raíz no es evidente.

Requerimientos identificados para un sistema gestor de fallas

Identificamos los siguientes requerimientos para un sistema de corrección de fallas y correlación de síntomas de manera automática:

- **Módulos para la comunicación a través de la red TCP/IP**: Nos permitirá comunicarnos con los distintos elementos de la red que utilizan éste protocolo.
- **Decodificación de mensajes de eventos**: Como los mensajes de eventos vienen codificados de acuerdo al protocolo que maneja cada elemento de red, éstos deben ser decodificados a un formato que permita su manejo interno por parte del sistema.
- **Almacenamiento de mensajes de eventos**: Dado que los recursos para procesar a los mensajes de eventos son compartidos y dado que la frecuencia con la que se generan los mensajes de eventos no es predecible, es necesario poder almacenar a estos mensajes de manera que se puedan procesar conforme se liberan estos recursos.
- **Identificación y categorización de mensajes de eventos**: Se necesita un proceso intermedio entre el proceso de tomar al mensaje del evento del contenedor y su procesamiento, para que el mensaje sea dirigido al proceso adecuado.
- **Representación del conocimiento para la corrección de fallas**: Es importante contar con un formato adecuado para la representación del conocimiento que permita la corrección de las fallas, este formato debe permitir ser leído y comprendido de forma que el sistema pueda construir el proceso al que representa.
- **Representación del conocimiento para la correlación de eventos**: Es importante contar con un formato que permita la codificación del conocimiento de manera que sea útil para realizar la identificación, diagnóstico y correlación de eventos.
- **Procesamiento de eventos mediante el uso de unidades de procesamiento**: Serán necesarias unidades de procesamiento que permitan la comunicación con los elementos de la red, así como la selección de acciones correctivas a partir de la información que se obtenga del estado de los elementos de la red relacionados con el mensaje de falla.
- **Correlación de eventos mediante el uso de sistemas expertos**: Los sistemas expertos son excelentes herramientas dentro del ámbito de la inteligencia artificial para realizar diagnósticos de manera automática. La estrategia es, que las unidades de procesamiento ofrezcan información relevante en el tratamiento de los mensajes de eventos para que permitan identificar fallas que no necesariamente son explícitas pero que pueden ser identificadas a

través de distintos síntomas colaterales.

- **Paralelización del procesamiento de mensajes de eventos**: Los sistemas de cómputo actuales permiten el procesamiento de manera simultánea, ésta cualidad la debemos aprovechar para procesar distintos mensajes de eventos al mismo tiempo.
- **Administración remota de servicios**: El sistema deberá poderse manejar de manera remota, pues los administradores de la red y las computadoras que administran a la red de telecomunicaciones, no siempre se encuentran en el mismo lugar.
- **Gestor de Notificaciones**: Son necesarias las notificaciones acerca del estado del sistema de corrección fallas y correlación de síntomas, para poder dar una monitorización adecuada con respecto al buen funcionamiento del mismo.

Estado del arte en la gestión automática de fallas

Artículos de investigación y publicaciones

La organización más común de arquitecturas para la administración de elementos de red, consiste en tener sistemas gestores de elementos o *Element Management System EMS* seguidos de Sistemas gestores de red o *Network Management System NMS* y dominados por una capa de Gestión de gestores o *Manager of Managers MOM*. Esta última capa es la responsable de realizar la correlación entre los eventos de los elementos manejados por la capa más básica. Para poder automatizar el proceso de diagnóstico de fallas, se requiere de una correcta representación del conocimiento que poseen los técnicos expertos. Para ello, encontramos a los *Sistemas Expertos* como un paradigma de la inteligencia artificial especializado en la identificación y diagnóstico para la resolución de problemas dentro de un dominio específico. Sin embargo, también revisamos artículos dónde se presenta la utilización de otros paradigmas de la inteligencia artificial como sistemas basados en conocimiento, redes neuronales artificiales, sistemas expertos difusos, razonamiento basados en casos, agentes inteligentes y minería de datos.

Entre otras cosas, también identificamos la disyuntiva entre la centralización o descentralización en el procesamiento para la gestión de fallas. Una de las ventajas de la descentralización es que el procesamiento se puede distribuir para obtener un mejor rendimiento, sin embargo la estructura centralizada simplifica algunos aspectos de la administración y facilita el análisis de correlación global de eventos. Nosotros utilizaremos el modelo centralizado porque facilita al sistema experto la correlación de eventos al tener un fácil y rápido acceso a los datos de toda la red.

Patentes

Identificamos cinco documentos de patentes que de alguna forma desempeñan funciones similares, donde describen una invención especializada para el diagnóstico y corrección de fallas utilizando estructuras de datos dinámicas para almacenar los procesos de corrección de fallas. Métodos para recibir y procesar mensajes de falla de aprovisionamiento de servicios en una red de banda ancha, la invención de métodos y sistemas de gestión de redes con capacidades de control y administración en tiempo real del tráfico producido por la transmisión de multimedia en redes de telecomunicaciones.

Patentes dentro de arquitecturas descentralizadas en su administración, que incorporan agentes que monitorizan diferentes aspectos de la red y ejecutan acciones correctivas de configuración, desempeño y seguridad. En conclusión, las invenciones existentes no se sobreponen con este proyecto, porque está centrado en la administración de fallas en general dentro de una red de telecomunicaciones.

Mercado actual

Los equipos que adquieren los operadores de redes de telecomunicaciones, incluyen herramientas informáticas para la administración de la red. Estas herramientas poseen la ventaja de estar especialmente diseñadas para el manejo de los equipos del mismo proveedor. Las redes al crecer tienen la opción de adquirir todos sus equipos al mismo proveedor, o componer su red de manera mixta, quedando entonces con la necesidad de adquirir tantos gestores de red como proveedores diferentes a los que hayan adquirido sus equipos.

Entre las herramientas de gestión figuran: *OSS-RC* con *FMX* de Ericsson, *MOSCAD* con *NFM* de Motorola y *AMS* de Alcatel. Estas herramientas ofrecen limitaciones que no permiten lidiar con el flujo de fallas producido por una red mixta. Adicionalmente, no todos los gestores cuentan con un módulo inteligente que automatice la tarea de diagnóstico y corrección de fallas.

Propuesta de un sistema de corrección de fallas y correlación de síntomas

Habiendo estudiado e identificado los requerimientos, nuestra propuesta se centra en satisfacer los siguientes puntos:

- **Alta compatibilidad e independencia de plataforma**: Ésta es la primera necesidad a satisfacer, el diseñar un gestor que permita hacer la corrección de eventos y correlación de síntomas de manera automática y que sea útil para equipos de distintos proveedores.
- **Gran capacidad de recepción y procesamiento de mensajes de eventos**: Para utilizar mejor los recursos disponibles necesitamos poder recibir y procesar mensajes de eventos de manera simultánea, y para esto necesitaremos de un contenedor de mensajes de eventos y de un gestor de procesos.
- **Representación del conocimiento para la corrección de fallas y correlación de síntomas**: Es de gran importancia, contar con una adecuada representación del conocimiento que permitirá la corrección de fallas y la correlación síntomas.

Para lograr el cumplimiento de éstos puntos utilizaremos una arquitectura que se conformará de los siguientes módulos:

- **Receptor de mensajes de eventos**: La recepción de mensajes de eventos es complicada porque los mensajes no se generan con una periodicidad predecible. Ante esta situación, identificamos que la recepción de mensajes se tiene que realizar a través de un contenedor *FIFO* que permita su desborde a disco duro en caso de saturarse.
- **Procesador de mensajes de eventos en paralelo**: Gracias a la proliferación de arquitecturas multiprocesador, podemos realizar de manera eficiente el procesamiento de mensajes de

eventos de manera simultánea. Pero para que esto ocurra es necesario contar con un módulo dentro del sistema que lleve la administración de estos procesos.

- **Unidades de procesamiento**: Las unidades de procesamiento deben representar un flujo de preguntas y respuestas para una toma de decisiones en cuánto a que procesos correctivos utilizar. Dentro de éstas unidades de procesamiento, adecuaremos un canal de comunicación hacia un sistema experto, al cual alimentaremos con información que encontremos importante para la identificación de problemas complejos en donde la causa raíz de la falla no es explícita. A través de archivos *XML* haremos la representación de las interconexiones entre las unidades de procesamiento, así como la configuración de los comandos que se ejecutarán en cada una de éstas unidades, éste formato nos permitirá utilizar las distintas herramientas existentes para su edición y visualización.
- **Sistema experto**: Utilizaremos un sistema experto para el diagnóstico de fallas no explícitas, este sistema experto será alimentado por las unidades de procesamiento con información relevante para la identificación de éste tipo de fallas.

Análisis de la propuesta

La propuesta está basada en la generación de diagnósticos dentro de una arquitectura de dos procesos simultáneos:

- **Procesamiento local**: Este proceso atiende a las fallas con unidades de procesamiento que corrigen los problemas de manera local y aislada, y durante su ejecución van recolectando información útil para diagnósticos de fallas no explícitas, o para encontrar eventos que son causa raíz de fallas.
- **Procesamiento global**: Aquí es donde el sistema analiza con un sistema experto la información generada en el primer proceso y determina si existe una falla no explícita, o eventos que son causa raíz de fallas.

Procesar los mensajes de eventos de ésta manera permite distribuir la carga de trabajo de manera eficiente, pues el sistema experto consume muchos recursos a cada inserción de información, y así a través de la programación de unidades de procesamiento, se puede tratar a la mayoría de mensajes de eventos de manera económica y sólo insertamos la información necesaria al sistema experto para la correlación.

Las condiciones ideales en las cuales funciona mejor esta arquitectura, se dan cuando la relación del tiempo que requieren los mensajes de eventos para ser procesados y la llegada de los mensajes se encuentra equilibrada. Pues podría encontrarse el contenedor de mensajes de eventos con un número muy grande de mensajes en espera para su procesamiento. También es importante recordar que este sistema está pensado para encontrarse dentro de un *OSS* o un *MOM*, pero que las tareas de integración dentro de estos sistemas quedan fuera de los alcances de esta tesina. Por otro lado estamos partiendo de la idea de que se cuenta con el rol de un experto humano que se encargará de codificar el conocimiento de los técnicos encargados de la administración de la red, para así integrar los procedimientos de identificación y corrección al sistema.

Futuras líneas de investigación

Para ayudar a crecer al trabajo que aquí expongo, existen varias líneas, de entre las cuales está el estudio de un método para la integración del sistema dentro de un *OSS* o un *MOM*, por otro lado, también es necesario el diseño de un contenedor de mensajes de eventos que permita el ordenamiento de los mismos con respecto a su prioridad.

Para la maduración de este proyecto se necesita mucho trabajo con respecto a las interfaces gráficas, que si bien no son indispensables, mejoran en gran medida la experiencia en la programación de los procesos necesarios para la abstracción del conocimiento que proviene de los expertos y que también son necesarias para el mantenimiento de las mismos, pues por poner un ejemplo, editar a los procesos en modo de sólo texto no permite ver fácilmente las transiciones entre las unidades de procesamiento, ni a los comandos y sus parámetros.

Y por último pero no menos interesante, se antoja una línea de investigación para establecer una metodología en cuanto a que, por un lado se pueda integrar de manera eficaz el conocimiento actual de los técnicos encargados de la administración de la red y por otra parte, establecer una metodología que permita el uso de diversas tecnologías dedicadas a la minería de datos, para utilizarlas sobre los históricos de los mensajes de eventos y que de esta manera se puedan descubrir patrones de comportamientos que son difíciles de identificar en cuanto a eventos raíz que desencadenan fallas, para así evitar que se vaya propagando el error por la red desde una etapa temprana. Y para esto existen dos perspectivas, una es la que corresponde a las fallas que existen y se identifican como fallas raíz, pero también existen las fallas que no son notificadas y que requieren del análisis de los eventos, para realizar su diagnóstico e identificación de manera automática al no ser una falla explícita. Queda así claro que para el descubrimiento de fallas raíz, se requiere del uso de dos perspectivas y posiblemente de dos métodos diferentes, a modo de ser codificadas en el sistema para su identificación y tratamiento.

Conclusiones

Como podemos observar, existen muchas ofertas en el mercado para satisfacer las demandas de los diferentes usuarios de redes de telecomunicaciones. Sin embargo no existe una opción que permita hacer la gerencia de todos los elementos en una red mixta, es decir, cuando han sido distintos los proveedores de los equipos que la conforman.

Es fácil pensar que quizá dos o más sistemas se pueden complementar muy bien, cada uno manejando a sus elementos en una red mixta, sin embargo, además del incremento en el costo del mantenimiento y las licencias, caemos en cuenta de que necesitamos una manera de intercomunicar a estos gestores de distintos proveedores, para así, poder tener una visión global del estado de la red y hacer posible la correlación para la identificación de fallas raíz.

Con la arquitectura aquí expuesta, es posible realizar las tareas de corrección automática de fallas, así como la identificación de fallas raíz, no necesariamente explícitas, a partir del diagnóstico y relación de eventos.

Capítulo II

Gestión automática de fallas en una red de telecomunicaciones

Breve introducción a las redes de telecomunicaciones

Una red de telecomunicaciones es una infraestructura física que permite ofrecer servicios de comunicación entre sus elementos, permitiendo que varios elementos utilicen una misma red para establecer canales de comunicación entre ellos. Y para que esto ocurra, es necesaria una infraestructura tan grande como el número de elementos que requieren su intercomunicación, su extensión y las complejidades que requiera la red por sus particularidades. También es importante realizar distintos tipos de gestión, como lo son la gestión de: configuraciones, rendimiento, contabilidad, seguridad y fallas. En éste trabajo nos enfocaremos en la gestión de fallas, más específicamente en la gestión automática de fallas.

Dentro de una red de telecomunicaciones, circulan una gran cantidad de mensajes codificados a través del protocolo *TCP/IP*. Estos mensajes indican el estado operativo de la red misma y se les llama *eventos*, algunos de estos eventos pueden ser *advertencias* con respecto al funcionamiento de la red, o bien pueden estar reportando *fallas* de sus componentes.

Los eventos son capturados por un sistema de gestión de red denominado *OSS Operations Support Systems*, o Sistema de soporte a las operaciones, y cuando la red es grande estos *OSS* pueden ser gestionados a su vez por un *MOM Manager of Managers*, o Manejador de manejadores. Ésta estructura jerárquica permite que los mensajes de eventos de fallas puedan ser escalados dependiendo la complejidad relacionada a la corrección de la falla. Generalmente los *OSS* pueden decodificar a los eventos de la red para filtrarlos por irrelevantes y condensar a los redundantes, en algunos casos incluso pueden estar provistos por módulos que permiten el manejo de las fallas para su corrección, a estos módulos se les llama *FM Fault Managers*, o Manejadores de Fallas.

Los mensajes de evento de falla dentro de una red de telecomunicaciones, son avisos que se envían hacia un gestor de red, cuya finalidad es avisar acerca del mal funcionamiento en hardware o software dentro de alguno de los elementos de la red. Desde el gestor de red se pueden identificar algunos de estos problemas y corregirlos de manera remota. La identificación, filtrado y corrección de estos problemas puede realizarse por el gestor de red o por técnicos especializados, que pueden realizar la corrección de manera remota o presencial según se requiera.

La meta de la automatización en la corrección de fallas está en corregir todas las fallas y sólo reportar a los técnicos las fallas que no puedan resolverse de manera remota. Actualmente la complejidad de las redes y el uso de tecnologías diversas, requieren de una preparación muy importante por parte de los técnicos que difícilmente se puede codificar en algoritmos que permitan resolver todos los problemas que se presentan, sobre todo en aquellos problemas que no son frecuentes y de difícil diagnóstico.

La motivación del presente proyecto se encuentra en que dentro de las redes de telecomunicaciones, la gran mayoría de las fallas son frecuentes, lo que permite identificar y codificar procesos para realizar la

corrección de manera automática. Así mismo, existen técnicas de inteligencia artificial que logran identificar fallas raíz, es decir, se puede preparar a un sistema de software con la finalidad de encontrar fallas que están originando a otras, y así al corregir a ésta falla origen, la corrección se propaga hacia otras fallas. No siempre es fácil encontrar a las fallas raíz pues no siempre se presentan de manera explícita, lo que deja como propósito, la identificación y el reporte de este tipo de fallas a un sistema de inteligencia artificial.

La identificación y corrección de fallas raíz ofrece una gran ventaja, porque permite mitigar de manera más eficiente a las fallas que se encuentran relacionadas. En este trabajo nos enfocaremos en un sistema que nos permita realizar la corrección automática de fallas a través de éstas dos técnicas: la corrección de fallas de manera local y la correlación para la identificación de fallas raíz.

A continuación daremos una revisión muy breve acerca de los componentes y procesos que conforman a la infraestructura de una red de telecomunicaciones y su operación, con la finalidad de conocer mejor el contexto en el que nos encontramos para el desarrollo de un sistema de gestión automática de fallas.

Tipos de redes de telecomunicaciones

Las redes de telecomunicaciones se pueden clasificar en *Redes públicas* cuando se utilizan para ofrecer servicios de telecomunicaciones al público en general y en *Redes privadas* cuando se utilizan para uso personal, sin dar acceso a terceros. Así mismo, de acuerdo a su cobertura geográfica se pueden clasificar en:

- **LAN *Local Area Network***: *Red de área local*, enlazan computadoras instaladas en un edificio u oficina.
- **MAN *Metropolitan Area Network***: *Red de área metropolitana*, se extiende dentro de una misma ciudad.
- **WAN *Wide Area Network***: *Red de área amplia,* se clasifican del siguiente modo:
  - *Redes de cobertura urbana*: Se compone de redes metropolitanas.
  - *Redes metropolitanas*: Se compone de redes de cobertura urbana.
  - *Redes nacionales*: Se compone de redes metropolitanas.
  - *Red global de telecomunicaciones*: Se compone de redes nacionales.

Elementos de hardware que componen a las redes de telecomunicaciones

Generalmente, las redes de telecomunicaciones se componen físicamente de:

- ***Equipos terminales***: Equipos a través de los cuales se obtiene entrada a la red por medio de un canal de acceso.
- ***Enlaces***: Son canales físicos de comunicación entre nodos o equipos y se dividen en:
  - *Canales que guían las señales desde la fuente hasta el destino*: Como lo son el cable de cobre o coaxial y la fibra óptica.
  - *Canales que difunden la señal sin una guía*: Como ocurre en un canal de radio, las

microondas y los enlaces vía satélite.

- ***Nodos***: Son equipos que proveen enlaces físicos entre los canales de comunicación de la red y realizan las siguientes funciones:
  - *Establecimiento y verificación de un protocolo*: Un protocolo es un conjunto de reglas que permiten la comunicación entre los nodos de la red.
  - *Transmisión*: Los mensajes se adaptan al canal, para su transporte eficiente y efectivo a través de la red.
  - *Interfase*: Proporcionando las señales de modo apropiado para ser transmitidas de acuerdo con el medio de que está formado el canal.
  - *Recuperación*: Cuando una transmisión se interrumpe, debe ser capaz de recuperarse y reanudar en cuanto sea posible la transmisión.
  - *Formateo*: Se necesita modificar el formato del mensaje para que pueda transitar entre redes que manejan distintos protocolos.
  - *Enrutamiento*: Cada nodo debe buscar la ruta más rápida para el envío de cada mensaje, y para esto toma en cuenta la congestión de la red.
  - *Repetición*: Existen protocolos que detectan si ha habido algún error en la transmisión y si es así puede solicitar una retransmisión.
  - *Direccionamiento*: Es la capacidad de identificar direcciones para poder hacer llegar un mensaje a su destino.
  - *Control de flujo*: Debe tener la capacidad de negar el envío de mensajes cuando el canal se encuentre saturado.

Composición funcional de las redes de telecomunicaciones

Diversos elementos de software integran funcionalmente a las redes de telecomunicaciones, entre los más comunes encontramos:

- **OS *Operating Systems***: Sistemas de operación, realizan las funciones de un gestor de red.
- **DCN *Data communication Network***: Redes de comunicación de datos, transportan la información de gestión.
- **WS *Workstations***: Estaciones de trabajo, permiten a los operadores acceder a la información de gestión.
- **MD *Mediation Device***: Dispositivos de mediación, que realizan las funciones de conversión de protocolos, mensajes, direcciones, encaminamiento y procesamiento de la información.
- **NE *Network Element***: Elementos de red, que se componen de elementos de la red de telecomunicación y un agente de gestión.
- **QA *Q Adapter***: Adaptadores Q, convierten datos de entrada y salida a un formato compatible con la red.
- **NOC *Network Operations Center***: Centro de Operaciones de Red, gestiona la red y ofrece información sobre la disponibilidad actual, histórica y planeada de los sistemas, el estado de la red, estadísticas de operación, la monitorización y la gestión de fallas.

Proceso de interconexión entre dos equipos de comunicación móvil

El proceso que ocurre al realizar una llamada entre dos dispositivos móviles es el siguiente:

1. El usuario inicia una llamada mediante su terminal que podría ser un teléfono móvil.
2. Se realiza un proceso de búsqueda por el canal de control para identificar un canal con buena recepción. Generalmente éste está asignado a la base más cercana. Esta búsqueda es controlada por el equipo móvil y se realiza cuando el equipo no se está utilizando en una conversación. Si durante la conversación una unidad en movimiento detecta que ha salido de la zona de cobertura, el sistema le asigna a esta conversación un nuevo canal y se realiza la nueva asignación sin que el usuario se percate de ello.
3. Una vez identificado el canal que será utilizado, la unidad móvil se considera inicializada y lista para establecer una comunicación.
4. Se envía el número seleccionado hacia la estación base, misma que envía esta información a la unidad de conmutación, que es la encargada de localizar a la unidad buscada. La trayectoria depende de la disponibilidad instantánea de los canales entre las centrales. Cuando la llamada se origina en un aparato de la red, se hace llegar la solicitud a la central celular de conmutación, la cual se encarga de localizar al usuario destino y de hacer la señalización correspondiente.
5. Una vez identificado el destino, se asigna un canal a la llamada y se notifica al usuario destino por medio de una señal de timbre para iniciar la conversación.
6. Si la línea del usuario destino está ocupada, la central lo detecta y le envía al usuario fuente la señal de ocupado.
7. Al terminar una conversación, ambos usuarios liberan los canales asignados para esa conversación y las unidades móviles reactivan la monitorización de la calidad de los canales.
8. Por último se contabiliza el costo de la llamada para su facturación.

Gestión de redes de telecomunicaciones

La gestión de redes incluye el despliegue, integración y coordinación del hardware, software y los elementos humanos para monitorizar, probar, sondear, configurar, analizar, evaluar y controlar los recursos de la red, para satisfacer los requerimientos de tiempo real, en el desempeño operacional y la calidad de servicio a un precio razonable.

En cuanto a la gestión de la red puede dividirse en capas:

- ***Capa de elementos de red***: Son los componentes básicos de la red, instalados como dispositivos físicos, capaces de distribuir datos y proveer de medios para ser controlados por el sistema de gestión.
- ***Capa de gestión de elementos de red***: Dentro de sus funciones básicas está la gestión de fallas, la información acerca del tráfico de datos y reunir información acerca del estado de cada elemento de la red.
- ***Capa de gestión de red***: Tiene la responsabilidad de gestionar la red en su totalidad. Ofrecer la provisión, cese o modificación de las capacidades de la red para el soporte de servicios a clientes, controla y coordina a todos los elementos de la red con su ámbito y dominio. Entre sus tareas más importantes se encuentran:

    - **Gestión de seguridad:** Para controlar el acceso a los recursos de la red de acuerdo a políticas bien definidas, así como el uso periódico de herramientas para analizar y controlar el uso legítimo de la red.
    - **Gestión de rendimiento:** Para garantizar niveles satisfactorios en el servicio a través de el establecimiento de niveles deseables y umbrales límite de rendimiento, una colección de datos suficientemente completa y organizada (estadísticas, tráfico, tasas de error, utilización, disponibilidad y capacidad de las instalaciones, etc) y una unidad de análisis de datos (humana o automática) que permita evaluar el rendimiento, e incluso pronosticarlo.
    - **Gestión de configuraciones:** Para mantener la información relativa al diseño de la red y su configuración actual, de entre las cuales encontramos:
        - ***Gestión de la Topología***: Que puede ser estática o dinámica, se necesita saber que elementos se encuentran instalados, dónde y cómo. Así como la información sobre las personas que responden por esos equipos.
        - ***Gestión de inventario y mantenimiento de Directorios***: Para la correcta administración de la red, se debe contar con una base de datos actualizada con los nombres de dominio, nodos y aplicaciones, así como un histórico de cambios y problemas.
        - ***Gestión de protocolos para el control operacional de la red***: El uso de protocolos simples de gestión de red (*SNMP Simple Network Management Protocol*) y accesos fuera de banda (*OOB Out of Bounds*) sirven para realizar diversas tareas como obtener el estado de un dispositivo, alterar su configuración, actualizar su software, abrir o cerrar sus puertos, reiniciarlos, entre otros.
- ***Capa de gestión de servicios***: Responsable de los aspectos contractuales de los servicios provistos a los clientes. Entre sus tareas están:
    - Interfaz con los clientes y otras administraciones.
    - Interacción con proveedores de servicios.
    - Mantenimiento de los acuerdos de nivel de servicio.
    - Mantenimiento de datos estadísticos.
    - Establecer relaciones entre servicios provistos por la red.
    - Gestión de contabilidad de los servicios de acceso, estadísticas de interfaces y protocolos.
- ***Capa de gestión comercial***: Tiene que ver con los aspectos técnicos y de negocio. Entre sus funciones están:
    - Gestión y responsabilidad comercial.
    - Tareas de asignación de objetivos.
    - Tarificas, contabilidad y control de gastos.
    - Control de inventario y nuevos servicios.

El problema de la gestión de fallas

Dentro de una red de telecomunicaciones, circulan una gran cantidad de mensajes que indican el estado operativo de la red misma, a estos mensajes se les llama *Eventos*, algunos de estos eventos pueden ser *Alarmas*, éstas alarmas pueden estar reportando una advertencia con respecto al funcionamiento de la red, o bien pueden estar reportando la *Falla* de alguno de sus componentes.

Los eventos son capturados por un sistema de gestión de red denominado *OSS Operations Support*

Systems, o Sistema de soporte a las operaciones, y cuando la red es grande estos *OSS* pueden ser gestionados a su vez por un *MOM Manager of Managers*, o Manejador de manejadores. Ésta estructura jerárquica permite que los mensajes de eventos de falla puedan ser escalados dependiendo la complejidad relacionada a la corrección de la falla. Generalmente los *OSS* pueden filtrar mensajes de eventos irrelevantes y condensar mensajes de eventos redundantes, en algunos casos incluso pueden estar provistos por módulos que permiten el manejo de las fallas para su corrección, a estos módulos se les llama *FM Fault Managers*, o Manejadores de Fallas. Dado que no todos los *OSS* cuentan con un *FM*, se advierte la necesidad de un módulo para el manejo de fallas que pueda ser útil dentro de las distintas tecnologías que pueden conformar a una red de telecomunicaciones. Para el desarrollo de un sistema para la gestión automática de fallas, estudiaremos las características de una falla dentro de una red de telecomunicaciones y los métodos para resolverlas.

Para una adecuada gestión de fallas, el procedimiento a seguir es el siguiente:

1. ***Identificación***: Algunas fallas son identificadas mediante el sondeo regular de los elementos de la red.
2. ***Aislamiento***: Planear un procedimiento para evitar que la falla repercuta ampliamente, con el objeto de minimizar la afectación del servicio.
3. ***Reacción***: Determinar prioridades, el escalamiento técnico, la gestión y asignación de recursos.
4. ***Resolución***: Aplicar procedimientos o medidas correctivas.
5. ***Notificación***: Realizar las notificaciones correspondientes al resolver una falla.

Para manejar la gran cantidad de alarmas que ocurren dentro de una red, se realizan tres procesos de transformación:

1. ***Monitorizado de alarmas***: Toma el estado original de una alarma y lo cambia a monitorizada *Monitoring*. Ese estado puede ser activado o desactivado, al desactivar el monitorizado de una alarma, esta pasa a terminada *Clear*.
2. ***Filtrado de alarmas***: Establece periodos a partir de los cuales determina un filtrado a partir de tres estados: presente *Present*, intermitente *Intermittent* o terminada *Clear*.
3. ***Enmascaramiento de alarmas***: Previene el reporte de alarmas innecesarias al deshabilitarlas.

Algunos métodos utilizados para la gestión de fallas son los siguientes:

- Automatizar las tareas.
- Mecanismos de reporte.
- Instalación y control de procedimientos de alarmas.
- Procedimientos de reparación y recuperación.
- Sistema de manejo de incidencias *Ticketing System*, que permite entre otras cosas dar seguimiento al estado actual de las fallas, personal asignado, tiempo estimado de soluciones, tiempos de trabajo e historial de acciones.
- Herramientas de monitorización como *ping*, *traceroute*, *ethereal/wireshark*, *snmp, HP Openview*, *Nagios* o *Big Brother*.
- Reportes de estado acerca de los nodos no-operativos *down* y nodos no alcanzables *unreachable.*

El problema de la correlación de fallas

Cuando el número de sus componentes, extensión y complejidad de las redes de telecomunicaciones aumentan, se reduce la capacidad de los administradores para resolver las fallas de manera rápida y eficaz. Por ello es deseable poder contar con procedimientos que permitan automatizar la corrección del mayor número de incidencias, de ahí que surge la necesidad de organizarlas por prioridad y atenderlas de acuerdo a esa prioridad. A partir de un proceso de correlación, la prioridad de una alarma puede cambiar de entre *Crítica*, *Mayor* o *Menor*. Las alarmas están relacionadas cuando la falla de un solo elemento de la red desencadena numerosos eventos de falla. El corazón del proceso de correlación de eventos es la determinación de las causas.

Para el manejo de fallas, existe una técnica que se enfoca en encontrar la falla causa raíz dentro de una serie de fallas que se utilizan a manera de síntomas. Su efectividad reside en que, al encontrar y corregir la falla raíz de otras fallas, el número total de fallas a atender se reduce considerablemente, incrementando la calidad en el diagnóstico y en la generación de estrategias para mantener en condiciones óptimas a los servicios provistos por la red de telecomunicaciones.

Monitorizar, filtrar y enmascarar eventos, reducen su cantidad, lo cual es vital, pero no encuentran las causas. Es importante encontrar las causas y para ello se necesita de un experto que determine las causas a partir de un conjunto de eventos, es ahí cuando herramientas de inteligencia artificial y minería de datos adquieren un papel importante para la asistencia en la determinación de las causas.

Es importante mencionar que no todas las fallas que son causa raíz de otras fallas, son explícitas, es decir, el elemento de red no siempre realiza la generación del mensaje de falla que refiere a la falla raíz que el mismo origina. Esto hace necesaria la creación del mensaje de falla a partir de un diagnóstico por parte de un especialista o por parte de un sistema automático de inteligencia artificial dotado con el conocimiento necesario para identificar la falla.

Automatización para la corrección de fallas y correlación de eventos

Dado que los elementos de la red tienen una serie de fallas bien identificadas, es muy útil la creación de procedimientos específicos para la corrección de fallas frecuentes. Algunos de estos procedimientos pueden ser aplicados de manera automática por parte de los *FM* dentro de los *OSS*. Manejar las fallas de manera automática ofrece muchos beneficios, de entre ellos está la reducción de fallas que deben ser atendidas por especialistas dentro del centro de servicio y también se encuentra una disminución del tiempo en la velocidad de respuesta.

Una manera de automatizar la corrección de fallas, es a través del uso de máquinas de estados, que como veremos más adelante, permiten procesar las fallas mediante un flujo de preguntas y respuestas a los elementos relacionados, así como la ejecución de procedimientos o *scripts* correctivos.

Así mismo, es muy conveniente la creación de procesos que permitan la identificación automática de fallas causa raíz a partir de una serie de síntomas de la red, pues al corregir éstas fallas el número de fallas total se reduce considerablemente, incrementando la calidad en el diagnóstico y en la generación de estrategias para la corrección de las fallas. Es importante mencionar que no todas las fallas que son

causa raíz de otras fallas, son explícitas, es decir, el elemento de red no siempre se realiza la generación del mensaje de falla que refiere a la falla raíz.

El propósito de un sistema experto es el de imitar la toma de decisiones que habría tomado un experto para realizar un diagnóstico y la aplicación de soluciones. Es por esto, que el sistema necesita de acceso a toda la información pertinente a través de preguntas y respuestas hacia los elementos de la red y a aquellos que se le relacionan para su correcto funcionamiento para incrementar el conocimiento que tiene acerca de una falla. Con un diagnóstico adecuado el sistema puede decidir entre solucionar la falla, reducir su impacto o simplemente ignorarlo. Cuando un elemento de red envía un mensaje de falla, este mensaje debe llegar al sistema experto en caso de ser útil en la identificación de fallas que se propagan por la red. Para el manejo de fallas en una red, necesitamos tener procedimientos para realizar un diagnóstico, para efectuar la corrección y para dar aviso de una falla que no pudo ser corregida.

Automatización mediante el uso de la inteligencia artificial

Estudiaremos brevemente algunos puntos fundamentales para el uso de las herramientas de inteligencia artificial, primero revisaremos algunos conceptos, para después establecer una serie de pasos importantes para la determinación y aplicación de técnicas de inteligencia artificial para la automatización en la correlación de eventos y corrección de fallas.

Aunque el concepto de inteligencia artificial no es muy claro ni fácil de definir, podemos comenzar pensando, que la inteligencia es la capacidad que tiene una entidad con respecto a la toma de decisiones en torno a la satisfacción de un conjunto de propósitos, que implican la resolución total o parcial de una serie de problemas. Con respecto a artificial, podemos entender que nos referimos al artificio realizado por un artesano, es decir, un producto manufacturado como lo puede ser la tela de una araña o el nido de algún ave, que es un trabajo artesanal que requiere del uso del intelecto para su realización. Bajo este contexto, para poder diferenciar de entre la inteligencia y la inteligencia artificial, podemos pensar en que la primera es resultado de un proceso evolutivo y la segunda es resultado de una manufactura en la que hubo alguna intervención intelectual. Pero hay que dejar claro que la inteligencia artificial no es similar a la humana, porque cuando hablamos del razonamiento de una máquina no pensamos en un razonamiento consciente, sino más bien en un proceso mecánico que da la apariencia de comprender el problema.

Herramientas de inteligencia artificial aplicada

De entre las técnicas comunes de inteligencia artificial aplicada en la solución de problemas encontramos:

- **Máquinas de estados**: Son una serie de estados relacionados por transiciones, en donde cada estado representa una situación particular del problema y las transiciones a una serie de alternativas para operar en la resolución del problema. Muy útil para la representación de situaciones predecibles.
- **Algoritmos genéticos**: Utilizan un proceso de evaluación, selección, cruza y mutación para generar, evaluar y encontrar soluciones. Muy útil para la búsqueda en un espacio de soluciones,

en donde la solución se encuentra dentro de una cuenca de atracción al que la población irá convergiendo paulatinamente.

- **Redes neuronales artificiales**: Utiliza un modelo matemático basado en las neuronas biológicas. Muy útiles para el reconocimiento de patrones en el caso del *Perceptrón* y *Madaline*, y también útiles para encontrar patrones dentro de una serie de datos como en el uso del *SOM Self Organized Maps*.
- **Redes bayesianas**: Propone soluciones mediante inferencia probabilística. Muy útiles en el tratamiento de problemas en donde la certidumbre está disminuida y se cuenta con un sistema de creencias.
- **Minería de datos**: Su propósito se cumple al encontrar patrones dentro de grandes volúmenes de datos. Muy útil para el estudio de datos en donde la probabilidad y la estadística se ven limitados en la búsqueda de patrones y tendencias.
- **Sistemas expertos:** Los sistemas expertos son una rama de la inteligencia artificial, donde el poder de resolución de un problema mediante un programa de computadora proviene del conocimiento de un experto sobre un dominio específico. Los sistemas expertos siguen ciertas reglas o pasos comprensibles de manera que se pueda generar la explicación para cada una de estas reglas, que a la vez se basan en hechos. Así mismo deben ser capaces de generar nuevo conocimiento en base a razonamientos.

Proceso de integración de la inteligencia artificial a sistemas

El poder de la inteligencia artificial es limitado y para lograr que funcione correctamente se debe realizar un diseño adecuado. A continuación se exponen algunos pasos importantes a considerar en el diseño de un sistema que incorpore el uso de alguna técnica de inteligencia artificial:

1. **Identificar el problema**: Lo más importante para poder identificar el problema es determinar claramente lo que queremos resolver, esto con la finalidad de realizar un modelo en donde sólo tomemos en cuenta los factores relevantes a la hora de tomar decisiones que resuelvan el problema.
2. **Identificar las entradas**: Debido a que la inteligencia artificial es limitada, debemos tomar especial cuidado en sólo alimentar al sistema inteligente con la información pertinente.
3. **Identificar las salidas**: Las salidas de un sistema inteligente pueden ser direccionadas de manera que puedan resolver el problema, informar a la persona o sistema que tenga la habilidad de ejecutar la serie de instrucciones que puedan resolver el problema.
4. **Encontrar un paradigma de inteligencia artificial adecuado**: Existen gran cantidad de estrategias y métodos para abordar un problema mediante el uso de la inteligencia artificial, hasta ahora no existe una estrategia que sea la más eficiente en todo. Incluso podemos considerar en fragmentar el problema, de modo en que la solución se pueda efectuar a través de la unión de estrategias.
5. **Modelar el problema de acuerdo al paradigma de inteligencia artificial seleccionado**: Debemos contar con una capa que de formato a la información de entrada para que sea compatible con el método de inteligencia artificial que hemos seleccionado, y de igual manera colocar una capa que permita interpretar las salidas del sistema en instrucciones útiles para los mecanismos que se encuentran encargados de efectuar la solución del problema.

Como hemos visto, existen diversos métodos de inteligencia artificial y debemos seguir una serie de pasos para su integración al sistema de gestión de fallas, en este trabajo abordaremos a los *Sistemas Expertos* y a las *Máquinas de Estados* por su utilidad en el diagnóstico y la resolución de problemas respectivamente. En las siguientes páginas analizaremos cuales son sus características y expondremos su utilidad en la gestión de fallas automática dentro de una red de telecomunicaciones.

Máquinas de estados para su uso en la corrección automática de fallas

La gestión de fallas se lleva a través de la administración de instrucciones o *scripts* que realizan dos principales tipos de actividades, la primera consiste en la consulta de datos a los elementos de red y la segunda en la ejecución de acciones preventivas o correctivas. Tomando en cuenta esto, el proceso de automatización reside en la correcta selección y aplicación de estos *scripts*.

Daremos una breve introducción a algunos tipos de máquinas de estados que son de interés para el desarrollo de un corrector de fallas automático. Veremos la *Máquina de estados finitos o autómata finito* porque es el modelo fundamental de las máquinas de estados, al *Transductor de estados finitos* y a la *Máquina de Mealy* porque son un tipo de máquina de estados que nos permiten generar salidas en respuesta a las transiciones y, finalmente, a la *Máquina de estados finitos extendida* porque toman en cuenta, además de la entrada, a una condicional para realizar las transiciones. Todo esto nos será útil al considerar a un mensaje de falla como un símbolo que nos llevará a identificar el procedimiento a seguir y a la respuesta de un *script* de consulta como el símbolo que determinará el rumbo de la transición entre un estado y otro. Así, estos fundamentos nos serán muy útiles en el diseño de una máquina de estados que nos permita codificar la administración en la ejecución de *scripts* ante los mensajes de fallas en una red de telecomunicaciones. Podríamos pensar que es una desventaja el que una máquina de estados realice las tareas de manera secuencial, pero no es así, porque precisamente esto genera un ahorro en la comunicación hacia los elementos de la red, que solo darán respuesta a las preguntas necesarias para determinar la aplicación de los *scripts* correctivos.

Máquina de estados finitos o autómata finito

Es un modelo matemático que se compone de una 5-tupla (Q, Σ, q_0, δ, F) donde:

- Q: Conjunto finito de estados.
- Σ: Alfabeto finito, el cual es un conjunto de símbolos.
- $q_0 \in Q$: Estado inicial único.
- $\delta: Q \times \Sigma \rightarrow Q$: Función de transición.
- $F \subseteq Q$: Conjunto de al menos un estado final o de aceptación.

Funcionamiento:

- Se construye una cadena de entrada a partir de los símbolos del alfabeto.
- La máquina de estado recibe esta cadena para procesar cada uno de estos símbolos conforme al orden en que llegan.
- El modo de procesar cada uno de los símbolos se define a través de la función de transición y se

comienza este proceso a partir del estado *qo*.

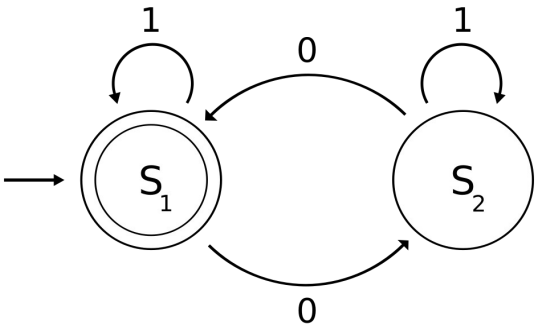
- La función de transición recibe uno a uno los símbolos a procesar y establece uno a uno los cambios de estado según se defina.
- Cuando todos los símbolos de la cadena de entrada han sido procesados y el autómata se encuentra en un estado final, se dice que la cadena ha sido aceptada por que pertenece al lenguaje reconocido por el autómata. En caso contrario, la cadena no pertenece este lenguaje.

Los autómatas finitos se pueden representar mediante grafos particulares, también llamados diagramas de estados finitos, de la siguiente manera:

- Los estados se representan como vértices, etiquetados con su nombre en el interior.
- Una transición desde un estado a otro, dependiente de un símbolo del alfabeto, se representa mediante una arista dirigida que une a estos vértices, y que está etiquetada con dicho símbolo.
- El estado inicial se caracteriza por tener una arista que llega a él, proveniente de ningún otro vértice.

Los autómatas finitos se pueden dividir en dos, según las características en su función de transición:

- **Autómata finito no determinista**: Puede tener definida una o más transiciones para cada pareja de estado y símbolo. Pueden definirse transiciones sin necesidad de leer algún símbolo, que se denotan como: *δ(q,ε)*. Cabe mencionar que estas características permiten a el autómata estar en varios estados a la vez.
- **Autómata finito determinista**: Es un tipo de autómata finito, que puede tener definida a lo más una transición para cada pareja de estado y símbolo. Sólo permite transiciones del estilo *δ(q,ε)* para los estados finales.



Este autómata finito está definido sobre el alfabeto *Σ = {0,1}*, posee dos estados *s1* y *s2*, y sus transiciones son *δ(s1, 0) = s2*, *δ(s1, 1) = s1*, *δ(s2, 0) = s1* y *δ(s2, 1) = s2*. Su estado inicial es *s1*, que es también su único estado final.

Transductor de estados finitos

Un transductor de estados finitos, o transductor finito, es un autómata finito con una cinta de entrada y otra de salida. Podemos decir que este tipo de autómatas traducen el contenido de la cadena de entrada en su cinta de salida. Esta transducción se puede realizar de forma no determinista y entonces se producirá más de una salida por cada cadena de entrada. Cuando un transductor no produce ninguna salida, se dice que el transductor rechaza la entrada.

Formalmente un transductor de estados finitos T es una 6-tupla $(Q, \Sigma, \Gamma, I, F, \delta)$ tal que:

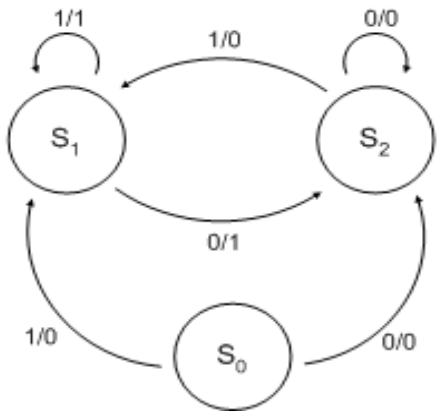
- Q: Es un conjunto finito, el conjunto de estados.
- Σ: Es un conjunto finito, llamado el alfabeto de entrada.
- Γ: Es un conjunto finito, llamado el alfabeto de salida.
- I: Es un subconjunto de Q, el conjunto de estados iniciales.
- F: Es un subconjunto de Q, el conjunto de estados finales.
- ε: Es la cadena vacía.
- $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \times Q$: Es la función de transición.

Se puede ver (Q, δ) como un grafo dirigido etiquetado, conocido como el grafo de transición de T. El conjunto de vértices es Q, y $(q, a, b, r) \in \delta$ indica que hay una arista etiquetada que va del vértice q al vértice r. También se dice que a es la etiqueta de entrada y b la etiqueta de salida de esa arista.

Máquina de Mealy

Formalmente una máquina de Mealy es una 6-tupla, $(Q, q_0, \Sigma, \Gamma, \delta, G)$:

- Q: Es un conjunto finito, el conjunto de estados.
- $q_0 \in Q$: Estado inicial único.
- Σ: Es un conjunto finito, llamado el alfabeto de entrada.
- Γ: Es un conjunto finito, llamado el alfabeto de salida.
- $\delta \subseteq Q \times \Sigma \rightarrow Q$: Es la función de transición.
- $G \subseteq Q \times \Sigma \rightarrow \Gamma$: Es la función de salida.



Esta máquina de Mealy está definida sobre el alfabeto $\Sigma = \{0,1\}$, posee tres estados *s1*, *s2* y *s3*. Sus transiciones δ y sus salidas G se definen a continuación:

$\delta(s0, 0) = s1$	$G(s0, 0) = 1$
$\delta(s0, 0) = s2$	$G(s0, 0) = 0$
$\delta(s1, 1) = s1$	$G(s1, 1) = 1$
$\delta(s1, 0) = s2$	$G(s1, 0) = 1$
$\delta(s2, 0) = s1$	$G(s2, 0) = 1$
$\delta(s2, 0) = s2$	$G(s2, 0) = 0$

Su estado inicial es *s1*, que es también su único estado final.

Máquina de estados finitos extendida

Formalmente una máquina de estados finitos extendida es una 7-tupla, (Σ, Γ, Q, D, F, U, T) donde:

- Q: Es un conjunto finito, el conjunto de estados.
- Σ: Es un conjunto finito, llamado el alfabeto de entrada.
- Γ: Es un conjunto finito, llamado el alfabeto de salida.
- D: Es un espacio lineal n-dimensional $D1 \times \ldots \times Dn$.
- F: Es conjunto de funciones de habilitación $fi: D \rightarrow \{0,1\}$.
- U: Es conjunto de funciones de actualización $ui: D \rightarrow D$.
- T: Es una relación de transición $T: Q \times F \times \Sigma \rightarrow Q \times U \times \Gamma$.

En una máquina de estados finitos convencional, las transiciones son asociadas a un conjunto de condiciones booleanas de entrada y a un conjunto de funciones booleanas de salida. En una máquina de estados finitos extendida, la transición puede ser expresada por un estatuto condicional, que consiste en una serie de condiciones de activación. Si las condiciones de activación se han satisfecho, la transición se realiza, llevando a la máquina del estado actual al siguiente estado y realizando las operaciones definidas para los datos.

Sistemas expertos para su uso en la correlación automática de fallas

Un sistema experto será de utilidad en la correlación de eventos para la identificación de fallas, porque toma partes de información como hechos que almacena en una base de conocimientos, y se pueden programar reglas que al aplicarse puedan relacionar a estos hechos para relacionarlos. Para esto, partiremos del hecho de que los técnicos especializados saben cuáles son el conjunto de síntomas que determinan la identificación de una falla raíz.

Los sistemas expertos son una rama de la inteligencia artificial, donde el poder de resolución de un problema mediante un programa de computadora proviene del conocimiento de un experto sobre un dominio específico. Los sistemas expertos siguen ciertas reglas o pasos comprensibles de manera que se pueda generar la explicación para cada una de estas reglas, que a la vez se basan en hechos. Así mismo deben ser capaces de generar nuevo conocimiento en base a razonamientos.

Un Sistema Experto está conformado por [14]:

- ***Base de conocimientos***: Contiene conocimiento modelado extraído del diálogo con un experto.
- ***Base de hechos o Memoria de trabajo***: Contiene el estado del problema sobre el cual se está trabajando.
- ***Motor de inferencia***: Modela el proceso de razonamiento.
- ***Módulos de justificación***: Explica el razonamiento utilizado por el sistema para llegar a una determinada conclusión.
- ***Interfaz de usuario***: Es la interacción entre el sistema experto y el usuario, se realiza mediante el lenguaje natural.

Principalmente existen tres tipos de sistemas expertos:

- **Basados en reglas previamente establecidas**: Aplicando reglas heurísticas apoyadas generalmente en lógica difusa para su evaluación y aplicación.
- **Basados en casos**: Aplicando el razonamiento basado en casos, donde la solución a un problema similar planteado con anterioridad se adapta al nuevo problema.
- **Basados en redes bayesianas**: Aplicando estadística y el teorema de Bayes.

Un sistema experto está conformado por dos procesos fundamentales:

- **Simulación**: Consistente en crear modelos basados en hechos, observaciones e interpretaciones, a fin de estudiar el comportamiento de los mismos mediante la observación de las salidas para un conjunto de entradas.
- **Control**: Realiza tareas de interpretación, diagnóstico y reparación de forma secuencial.

Por lo tanto, el tipo de sistema experto que nos conviene es el basado en reglas previamente establecidas, en donde representemos el conocimiento de los expertos en reglas que nos permitan identificar y reportar fallas que se encuentren relacionadas en el reconocimiento de una falla raíz explícita o implícita.

Monitoreo tecnológico

Identificamos cinco documentos de patentes que de alguna forma desempeñan funciones similares:

EP2109323-A1/B1	Dynamic fault analysis for a centrally managed network element in a telecommunications system
US20110099428-A1	Dynamic fault analysis for a centrally managed network element in a telecommunications system
GB2412539-A	Processing fault reports and generating corrective solutions in a broadband communications network
US2010/0318652-A1	Apparatus and methods for real-time multimedia network traffic management & control in wireless networks
EP0743778-B1	Service and information management system for telecommunications network

Los documentos *US20110099428-A1*, *EP2109323-A1*, *EP2109323-B1*, describen una invención especializada en el diagnóstico y corrección de fallas en una estación base, utilizando estructuras de datos dinámicas para almacenar procedimientos correctivos. El documento *GB2412539-A* describe métodos para recibir y procesar mensajes de eventos de aprovisionamiento de servicios en una red de banda ancha. Por su parte la patente *US0318652-A1*, que es la más cercana a este proyecto, reclama la invención de métodos y sistemas de gestión de redes con capacidades de control y administración, en tiempo real, del tráfico producido por la transmisión de multimedia en redes de telecomunicaciones. La arquitectura propuesta descentraliza la administración incorporando agentes que monitorizan diferentes aspectos de la red y ejecutan acciones correctivas, de configuración, desempeño y seguridad. Por último el documento *EP0743778 B1* describe un equipo que provee el servicio de llamadas telefónicas. En conclusión, las invenciones existentes no se sobreponen con este proyecto. La patente *US0318652-A1*, que es la más cercana, está orientada a la administración del tráfico en una red multimedia, mientras que el proyecto está centrado a la administración de fallas en general, dentro de una red de telecomunicaciones.

Artículos de investigación y publicaciones

En esta sección referimos algunos artículos en los que se discuten algunos de los asuntos técnicos que nosotros consideramos que deben ser tomados en cuenta para el diseño y desarrollo de un sistema inteligente como el que aquí proyectamos.

El problema de gestionar automáticamente las fallas de una red es ampliamente reconocido y ha sido abordado con muchos enfoques diferentes. La *International Telecommunications Union ITU* reconoce la relevancia de este problema e incluso recomienda una clasificación para el conjunto de acciones que deben realizarse para atender las fallas [1].

La organización más común de arquitecturas para la administración de elementos de red, consiste en tener sistemas gestores de elementos o *Element Management System EMS* seguidos de Sistemas gestores de red o *Network Management System NMS* y dominados por una capa de Gestión de gestores o *Manager of Managers MOM*. Esta última capa es la responsable de realizar toda la correlación entre los eventos de los elementos manejados por la capa más básica [30].

Para poder automatizar el proceso de diagnóstico de fallas, se requiere de una correcta representación del conocimiento que que poseen los técnicos expertos [7, 10]. Para ello, los *Sistemas Expertos* son la rama de la inteligencia artificial especializada en la identificación y diagnóstico para la resolución de problemas dentro de un dominio específico, porque dentro de una red de telecomunicaciones se conocen los elementos que la conforman y los procedimientos específicos para la identificación y corrección de fallas. Por ello, la tecnología de los sistemas expertos ha sido utilizada extensivamente para la gestión de fallas en este tipo de redes [2]. Sin embargo otros paradigmas emergentes de la inteligencia artificial están siendo utilizados: sistemas basados en conocimiento, redes neuronales artificiales, sistemas expertos difusos, razonamiento basados en casos, agentes inteligentes, minería de datos, entre otros [3, 4, 5, 11, 12, 13].

Cuando se trata de administrar redes de gran escala, es importante que el diseño del sistema inteligente tenga la capacidad de procesar y realizar el diagnóstico de un flujo intenso de mensajes de eventos, lo cual requiere un alto nivel de desempeño para que el procesamiento se pueda llevar a cabo en tiempo real [6, 7]. Ante este reto, emerge la disyuntiva de la descentralización del procesamiento contra la concentración del mismo en un módulo que centralice el procesamiento. Una de las ventajas de la descentralización es que el procesamiento se puede distribuir para obtener un mejor rendimiento [8, 9, 10], sin embargo la estructura centralizada simplifica algunos aspectos de la administración y facilita el análisis de correlación de eventos de toda la red. Nuestra solución busca centralizar el diagnóstico y corrección de fallas.

Sistemas de corrección de fallas y correlación de eventos de manera automática

Las tecnologías que adquieren los operadores de telefonía móvil incluyen como valor agregado herramientas informáticas para la administración de la red. Generalmente el operador adquiere estas herramientas para garantizar el buen funcionamiento de los equipos. Al final del día termina con subredes de diferentes tecnologías administradas cada una por su propio gestor. Entre las herramientas

de gestión figuran: *OSS-RC* con FMX de Ericsson, *MOSCAD* con NFM de Motorola y *AMS* de Alcatel. Adicionalmente, no todos las empresas proveedoras de tecnologías cuentan con gestores que contengan un módulo inteligente que automatice la tarea de diagnóstico y corrección de fallas.

OSS-RC con FMX de Ericsson: El *OSS-RC* de Ericsson es un gestor de red completo, creado para la infraestructura de red de Ericsson. Utilizado por operadores de todo el mundo, integra y gestiona una amplia gama de componentes de red. Utilizado junto con la oferta de banda ancha de Ericsson se obtiene una solución integral para la gestión total de la red de telecomunicaciones.

MOSCAD con NFM de Motorola: El sistema de manejo de fallas de red provee de un centro de control de redes que tiene acceso a la comunicación y al ambiente de dispositivos remotos. Esto le provee al operador y al administrador de sistema con algunas de las herramientas necesarias para identificar y hasta corregir fallas en la red de telecomunicación. El *Gateway IP* ofrece una interfaz estándar con una amplia gama de plataformas soportadas incluyendo *HP/OpenView*, *SUN/Solstice*, *IBM/NetView* y más.

AMS de Alcatel: El *AMS 5526 de Alcatel* provee de un manejo que permite monitorizar y proveer servicios *DSL*, datos y voz. El *AMS* cuenta con una interfaz gráfica que permite utilizar y aprender fácilmente. Soporta totalmente el manejo remoto, lo cual provee de la flexibilidad que requiere el manejo de múltiples plataformas geográficamente dispersas desde un solo lugar. Es escalable a hasta 2500 elementos de red y 100 elementos concurrentes.

A continuación se muestra un cuadro comparativo de las 3 tecnologías disponibles, listadas arriba:

Característica	**Ericsson**	**Motorola**	**Alcatel**
Número total de elementos que pueden manejarse en una red	*Sin límite*	*Sin límite*	*2,500*
Número de usuarios simultáneos que pueden manejar la red	*Sin límite*	*Sin límite*	*1,000*
Capacidades de correlación	*Si (FMX)*	*No*	*No*
Interfaz gráfica amigable	*Si*	*Si*	*Si*
Interoperabilidad con elementos de diferentes proveedores	*No*	*Si*	*Si*
Puede manejar una red completamente independiente de otros sistemas *OSS*	***No***	***No***	***No***

Como podemos observar en el cuadro anterior, existen algunas ofertas en el mercado para satisfacer las demandas de los diferentes usuarios de redes de telecomunicación. Sin embargo no existe una que permita hacer la gerencia de todos los elementos de una red.

Es fácil pensar que quizá dos o más sistemas se pueden complementar muy bien cada uno manejando los elementos de su empresa, sin embargo fácilmente caemos en cuenta que la correlación entre elementos es indispensable para un manejo eficiente y si los sistemas de diferentes proveedores no se pueden comunicar esto es prácticamente imposible.

Capítulo III

Propuesta de un sistema de corrección de fallas y correlación de eventos de manera automática para redes de telecomunicaciones

Análisis del problema

El objetivo de una red de telecomunicaciones es permitir la comunicación entre sus elementos, pero para que esto ocurra, se debe conformar una infraestructura que permita el envío del mensaje desde un punto de la red a otro. Esta infraestructura se compone de distintos elementos de red que permiten interconexiones temporales entre otros elementos que desean comunicarse entre sí, así como también se compone de elementos que permiten la monitorización de los nodos, para que en caso de que ocurra un problema en el hardware o software se haga un mensaje con la notificación correspondiente.

Dentro de una red de telecomunicaciones, circulan una gran cantidad de mensajes que indican el estado operativo de la red misma, a estos mensajes se les llama *eventos*, algunos de estos eventos pueden ser *advertencias* con respecto al funcionamiento de la red, o bien pueden estar reportando *fallas* en sus componentes.

Los mensajes de falla dentro de una red de telecomunicaciones, son avisos que se envían hacia un gestor de red, cuya finalidad es avisar acerca de el mal funcionamiento en hardware o software dentro de alguno de los elementos de la red. Desde el gestor de red se pueden identificar algunos de estos problemas y corregirlos de manera remota. La identificación, filtrado y corrección de estos problemas puede realizarse por el gestor de red o por técnicos especializados, que pueden realizar la corrección de manera remota o presencial según se requiera.

Por ahora, las fallas dentro de las redes complejas de comunicación son inevitables, pero se pueden identificar, aislar, reaccionar ante ellas, resolverlas y notificar su estatus final. Esto ocurre, porque cuando estas fallas son frecuentes, podemos encontrar un patrón que nos permita identificarlas y un procedimiento para resolverlas. Estos patrones y estos procedimientos pueden ser codificados de manera que puedan ejecutarse de manera automática. Para que esto ocurra necesitamos un sistema que nos permita recibir mensajes de eventos y darles un tratamiento clasificándolos dentro de una serie de patrones que corresponden a una serie de procedimientos que se encargarán de procesarlos. En caso de que el evento no pueda ser identificado por algún patrón o no pueda ser resuelto por ningún procedimiento, entonces se hará la notificación correspondiente al centro de servicio encargado de la administración de la red, para así atenderlo de manera personalizada. Todo esto lo haremos partiendo del hecho de que existen técnicos que se ocupan del mantenimiento de la red, que han identificado a estos patrones y a sus respectivos procedimientos. Como trabajo a futuro quedará el estudiar métodos de minería de datos que permitirán identificar de manera automática a los patrones para la categorización y posiblemente la generación automática de procedimientos correctivos.

Dentro de la infraestructura de una red de telecomunicaciones, se cuenta con un centro de servicio que se encarga de mantener a la red operando en condiciones óptimas. Cuando la red es grande y compleja, este centro de servicio se apoya de las herramientas de monitorización que ofrecen los *OSS* para

identificar los problemas y solucionarlos. Dado que los elementos de la red tienen una serie de fallas bien identificadas, es muy útil la creación de procedimientos específicos para la corrección de fallas frecuentes. Manejar las fallas de manera automática ofrece muchos beneficios, de entre ellos está reducir el número de fallas que deberán ser atendidas por especialistas dentro del centro de servicio y la disminución del tiempo en la velocidad de respuesta.

Los eventos son capturados por un sistema de gestión de red denominado *OSS Operations Support Systems*, o Sistema de soporte a las operaciones, y cuando la red es grande estos *OSS* pueden ser gestionados a su vez por un *MOM Manager of Managers*, o Manejador de manejadores. Ésta estructura jerárquica permite que los mensajes de eventos puedan ser escalados dependiendo la complejidad relacionada a su procesamiento. Generalmente los *OSS* pueden filtrar mensajes de eventos irrelevantes y condensar a los que son redundantes, en algunos casos incluso pueden estar provistos por módulos que permiten el manejo de las fallas para su corrección, a estos módulos se les llama *FM Fault Managers*, o Manejadores de Fallas. Dado que no todos los *OSS* cuentan con un *FM*, se advierte la necesidad de un módulo para el manejo de fallas que pueda ser útil dentro de las distintas tecnologías que pueden conformar a una red de telecomunicaciones.

La meta de la automatización en la corrección de fallas está en corregir todas las fallas y sólo reportar a los técnicos fallas que no puedan resolverse de manera remota. Actualmente la complejidad de las redes y el uso de tecnologías diversas, requieren de una preparación muy importante por parte de los técnicos que difícilmente se puede codificar en algoritmos que permitan resolver todos los problemas que se presentan, sobre todo en aquellos problemas que no son frecuentes y de difícil diagnóstico.

Es importante mencionar que no todas las fallas que son causa raíz de otras fallas, son explícitas, es decir, el elemento de red no siempre realiza la generación del mensaje de falla que refiere a la falla raíz que el mismo origina. Esto hace necesaria la creación del mensaje de falla a partir de un diagnóstico por parte de un especialista o por parte de un sistema automático de inteligencia artificial dotado con el conocimiento necesario para identificar la falla.

La identificación y corrección de fallas raíz ofrece una gran ventaja, porque permite mitigar de manera más eficiente a las fallas que se encuentran relacionadas. En este trabajo nos enfocaremos en un sistema que nos permita realizar la corrección automática de fallas a través de éstas dos técnicas: la corrección de fallas de manera local y la correlación e identificación de fallas raíz.

En el desarrollo de un sistema que cubra las necesidades antes expuestas, no abordaremos las dificultades que existen para integrarlo dentro de un *OSS* pues fundamentalmente sólo nos centramos en el diseño de una arquitectura que nos permite satisfacer la recepción masiva de mensajes de fallas, su procesamiento en paralelo y la relación de síntomas inconexos para realizar el diagnóstico de una falla cuya falla raíz no es evidente.

Requerimientos

Dentro de los requerimientos identificados para la corrección de fallas encontramos dos tareas primordiales:

1. **Identificación del tipo de falla:** Identificando ciertas características de la falla, se identifica a

que categoría pertenece la falla a través de distintos filtros.
2. **Ejecución del proceso de corrección:** Se utiliza el proceso de corrección acorde a la categoría de la falla identificada en el paso anterior, este proceso de corrección debe contener distintas etapas de diagnóstico que permitan lidiar con las particularidades de la falla y los elementos de red relacionados.

Para poder efectuar estos dos procesos de manera automática hemos identificado los siguientes requerimientos a cumplir:

Recepción de mensajes de eventos

El primer problema con el que nos encontramos es el de la recepción de mensajes de eventos, porque éstos no se generan con una periodicidad predecible. Entonces, así como puede no presentarse ningún mensaje en periodos largos, también pueden presentarse una gran cantidad de mensajes en lapsos muy cortos. Otro problema es que el procesamiento de los mensajes de eventos no es inmediato, algunos requieren de esperas, por ejemplo al reiniciar algún elemento de red se debe esperar algunos minutos hasta que el elemento se termine de apagar para luego iniciar nuevamente hasta alcanzar un estado operativo. Ante estas situaciones, identificamos que la recepción de mensajes de eventos se tiene que realizar a través de un contenedor *FIFO* que permita su desborde a disco duro en caso de saturarse. Esto permite al sistema atender a los mensajes de eventos conforme cuente con los recursos necesarios para hacerlo, sin dejar desatendida su recepción. Tampoco hay que obviar el hecho de que debe haber un módulo que sirva de interfaz con la red para recibir a estos mensajes.

Administración remota

Como el sistema y su administrador no se encuentran necesariamente en el mismo lugar, se requiere de una interfaz que permita administrar al sistema vía red, que permita acceder al estado de cada uno de los componentes y disponga de las operaciones necesarias para llevar acabo su operación.

Procesamiento de mensajes de eventos en paralelo

Gracias a la proliferación de arquitecturas multiprocesador, podemos realizar de manera eficiente el procesamiento de los mensajes de eventos de manera simultánea. Pero para que esto ocurra es necesario contar con un módulo dentro del sistema que lleve la administración de estos procesos. El módulo de gestión de procesos permite establecer un número máximo de procesos ejecutándose de manera simultánea, así como llevar el control de estos procesos para terminarlos en caso de ser necesario y ofrecer información relacionada con cada uno de los procesos con la finalidad de llevar a cabo su monitorización por parte de un técnico especialista.

Unidades de procesamiento

Para corregir una falla de manera automática se requiere de varios pasos, uno de ellos es la identificación de procesos que permitan corregir las fallas, otro paso es identificar el flujo de la

ejecución de una serie de preguntas y respuestas entre los elementos de la red relacionados con la falla y el sistema que realiza el diagnóstico de la causa de la falla, esto con la finalidad de identificar el origen de la falla y entonces seleccionar el proceso adecuado para llevar a cabo de manera automática la corrección de la falla. Las unidades de procesamiento deben representar ese flujo de preguntas y respuestas para una toma de decisiones en cuánto a que procesos correctivos utilizar. Para esto, cada unidad de procesamiento debe estar provista de mecanismos que permitan su comunicación con los elementos de la red. Esto se puede realizar a través de una interfaz hacia la línea de comandos del sistema anfitrión, pues la comunicación con los elementos de la red se puede llevar a cabo desde ahí. Dentro de éstas unidades de procesamiento, adecuaremos un canal de comunicación hacia un sistema experto, al cual alimentaremos con información que encontremos importante para la identificación de problemas complejos en donde la causa raíz de la falla no es explícita.

Codificación de los procedimientos de corrección de fallas

Para poder realizar la corrección de fallas, se requiere de la codificación de los procedimientos de manera que puedan ser representados adecuadamente, lo cual permitirá al sistema ejecutarlos mediante los comandos descritos en cada una de las fases del proceso. Todo esto lo haremos a través de archivos con el formato XML, que nos permite utilizar las distintas herramientas existentes para su edición y visualización.

Sistema experto

Utilizaremos un sistema experto para el diagnóstico de fallas no explícitas, este sistema experto será alimentado por las unidades de procesamiento con información relevante para la identificación de éste tipo de fallas. La generación de diagnósticos dentro de una arquitectura consta de dos procesos simultáneos: El primer proceso atiende a los mensajes de eventos con unidades de procesamiento de manera aislada, recolecta información que podría ser útil para diagnósticos complejos e intenta solucionar a las fallas de manera local. El segundo proceso analiza con un sistema experto la información generada en el primer proceso por diferentes mensajes y determina si existe una falla no explícita al relacionarlos. Procesar las fallas de ésta manera permite distribuir la carga de trabajo de manera eficiente, pues el sistema experto consume muchos recursos a cada inserción de información, y así a través de la programación de unidades de procesamiento, se puede tratar a la mayoría de mensajes de eventos de manera económica y sólo insertamos la información necesaria al sistema experto para la correlación.

Inteligencia artificial como herramienta de automatización

De entre las técnicas de inteligencia artificial aplicada en la solución de problemas encontramos útiles para nuestro problema a las siguientes:

- **Máquinas de estados:** Son una serie de estados relacionados por transiciones, en donde cada estado representa una situación particular del problema y las transiciones a una serie de alternativas para operar en la resolución del problema. Muy útil para la representación de situaciones predecibles.

- **Minería de datos:** Su propósito se cumple al encontrar patrones dentro de grandes volúmenes de datos. Muy útil para el estudio de datos en donde la probabilidad y la estadística se ven limitados en la búsqueda de patrones y tendencias.
- **Sistemas expertos:** Los sistemas expertos son una rama de la inteligencia artificial, donde el poder de resolución de un problema mediante un programa de computadora proviene del conocimiento de un experto sobre un dominio específico. Los sistemas expertos siguen ciertas reglas o pasos comprensibles de manera que se pueda generar la explicación para cada una de estas reglas, que a la vez se basan en hechos. Así mismo deben ser capaces de generar nuevo conocimiento en base a razonamientos.

En el presente trabajo, sólo utilizaremos a las máquinas de estados para la representación de procedimientos de corrección de fallas, a los sistemas expertos para correlacionar eventos, y la minería de datos se dejará como futura línea de investigación, para su uso como herramienta complementaria en la identificación de fallas de causa raíz.

Corrección automática de fallas con máquinas de estados

La gestión de fallas se lleva a través de la administración de instrucciones o *scripts* que realizan dos principales tipos de actividades, la primera consiste en la consulta de datos a los elementos de red y la segunda en la ejecución de acciones preventivas o correctivas. Tomando en cuenta esto, el proceso de automatización reside en la correcta selección y aplicación de estos *scripts*.

Todo esto nos será útil al tomar a un mensaje del evento como un símbolo que nos ayudará a identificar el procedimiento a seguir y a la respuesta de un *script* de consulta como el símbolo que determinará el rumbo de la transición de entre un estado y otro. Así, estos fundamentos nos serán muy útiles en el diseño de una máquina de estados que nos permita codificar la administración en la ejecución de *scripts* ante los mensajes de eventos en una red de telecomunicaciones. Podríamos pensar que es una desventaja el que una máquina de estados realice las tareas de manera secuencial, pero no es así, porque precisamente esto genera un ahorro en la comunicación hacia los elementos de la red, que solo darán respuesta a las preguntas necesarias para determinar la aplicación de los *scripts* correctivos.

Correlación automática de fallas con sistemas expertos

Las alarmas están relacionadas cuando la falla de un solo elemento de la red desencadena numerosos eventos de falla. Un sistema experto será de utilidad en la identificación de fallas de correlación porque toma a cada falla como un hecho que almacena en una base de conocimientos, y se pueden programar procesos que relacionen fallas entre sí. Partiremos del hecho de que los técnicos especializados saben cuáles son el conjunto de síntomas que determinan la identificación de una falla raíz. Principalmente existen tres tipos de sistemas expertos: basados en reglas previamente establecidas, basados en casos y los basados en redes bayesianas.

Por lo estudiado en el ***Capítulo II***, el tipo de sistema experto que nos conviene es el basado en reglas previamente establecidas, porque nos permite codificar el conocimiento de los expertos en reglas que permiten identificar y reportar fallas que se encuentran relacionadas en el reconocimiento de una falla

raíz explícita o implícita. El problema que tendríamos con un sistema basado en casos es que las operaciones entre elementos de la red son muy particulares y no le permitiría sacar el potencial de probar otras soluciones con otros elementos de la red para los cuales las reglas no fueron diseñadas, y el problema que tendríamos con un sistema experto basado en redes bayesianas es que los mensajes de falla dentro de la red son muy concretos y no se basan en probabilidades, lo cual deja fuera a la naturaleza de este tipo de sistemas.

Dentro de las funciones de un *OSS* están: monitorizar, filtrar y enmascarar eventos. Todo esto ataca la reducción en la cantidad de eventos, lo cual es vital, pero no encuentra las causas que permitirían reducir su volumen. Para una reducción eficiente de el volumen de fallas relacionadas, es importante encontrar la causa, y para ello se necesita de un técnico especialista que la identifique a partir de un conjunto de eventos, es ahí cuando las herramientas de inteligencia artificial y minería de datos tienen un papel importante para encontrar la causa raíz de eventos, pues permiten identificar y encontrar patrones dentro de grandes cantidades de datos.

Es importante mencionar que no todas las fallas que son causa raíz de otras fallas, son explícitas, es decir, el elemento de red no siempre realiza la generación del mensaje de falla que refiere a la falla raíz que el mismo origina. Esto hace necesaria la creación del mensaje de falla a partir de un diagnóstico por parte de un especialista o por parte de un sistema automático de inteligencia artificial dotado con el conocimiento necesario para identificar la falla.

Propuesta

A continuación estudiaremos una arquitectura que resuelve el problema antes descrito, en donde la recepción de mensajes de eventos se realiza a través del desarrollo de un contenedor *FIFO* que permite su desborde a disco duro, el procesamiento en paralelo se logra mediante el desarrollo de un gestor de hilos para el procesamiento de los mensajes de eventos y la generación de diagnósticos para la correlación mediante una arquitectura con dos procesos simultáneos: el primer proceso atiende a los mensajes de eventos de manera local con máquinas de estados que recolectan información que podría ser útil para los diagnósticos complejos, y el segundo proceso analiza toda la información generada por el primer proceso con un sistema experto para determinar si existe alguna falla de correlación.

En el desarrollo de un sistema que cubra las necesidades antes expuestas, no abordaremos las dificultades que existen para integrarlo dentro de un *OSS* pues fundamentalmente sólo nos centramos en el diseño de una arquitectura que nos permite satisfacer la recepción masiva de mensajes de eventos, su procesamiento en paralelo y la relación de síntomas inconexos para realizar el diagnóstico de una falla cuya falla raíz no es evidente. Así como trabajaremos bajo el supuesto de que las comunicaciones no necesitarán de códigos de seguridad entre el *OSS* y nuestro sistema, pues se encontrarán dentro de una red privada. Podría quedar entonces como trabajo a futuro el cifrado en las comunicaciones entre nuestro sistema y los *OSS* o el *MOM* en caso de querer descentralizar los módulos.

Es importante aclarar, que este sistema de inteligencia artificial requiere del rol de un experto humano, que se encargue de codificar el conocimiento de los técnicos encargados de la administración de la red, para así integrar los procedimientos de identificación y corrección al sistema. Se puede pensar a futuro en utilizar técnicas de minería de datos para la identificación de causas raíz con la finalidad de generar

los procedimientos de identificación de causas, pero los resultados de estos análisis deben estar sujetos al análisis del experto humano antes de integrarse al sistema en producción.

Puntos estratégicos

Habiendo identificado los requerimientos, nuestra propuesta se centra en satisfacer los siguientes puntos de manera fundamental:

- **Alta compatibilidad e independencia de plataforma:** Necesitamos poder comunicarnos con elementos de una red de proveedores mixta, para esto necesitamos software que sea independiente de plataforma y de mecanismos adecuados para la comunicación con elementos de diferentes tecnologías. Ésta es la primera necesidad a satisfacer pues es el área de oportunidad que encontramos y que sirvió de motivación para la realización de ésta tesina, el diseñar un gestor que permita hacer la corrección de fallas y correlación de eventos de manera automática y que sea útil para equipos de distintos proveedores.
- **Gran capacidad de recepción y procesamiento de mensajes de eventos:** Para utilizar mejor los recursos disponibles necesitamos poder recibir y procesar mensajes de eventos de manera simultánea, y para esto necesitaremos de un contenedor de mensajes y de un gestor de procesos.
- **Representación del conocimiento para la corrección de fallas y correlación de eventos:** Es de extrema importancia, contar con una adecuada representación del conocimiento que permitirá la corrección de fallas y la correlación de eventos, pues de ello depende directamente el cumplimiento del propósito de este sistema, sobre todo en la identificación de fallas que son causa raíz de otras fallas.

Módulos principales

Para lograr el cumplimiento de éstos puntos utilizaremos una arquitectura que se conformará esencialmente de los siguientes módulos:

- **Receptor de mensajes de eventos:** La recepción de eventos es complicada porque no se generan con una periodicidad predecible. Ante estas situaciones, identificamos que la recepción de mensajes de eventos se tiene que realizar a través de un contenedor *FIFO* que permita su desborde a disco duro en caso de saturarse.
- **Procesador de mensajes de eventos en paralelo:** Gracias a la proliferación de arquitecturas multiprocesador, podemos realizar de manera eficiente el procesamiento de las fallas de manera simultánea. Pero para que esto ocurra es necesario contar con un módulo dentro del sistema que lleve la administración de estos procesos.
- **Unidades de procesamiento:** Las unidades de procesamiento deben representar el flujo de preguntas y respuestas a los elementos de la red para una toma de decisiones en cuánto a que procesos correctivos utilizar.
- **Sistema experto:** Utilizaremos un sistema experto para el diagnóstico de fallas no explícitas, este sistema experto será alimentado por las unidades de procesamiento con información relevante para la identificación de éste tipo de fallas.

Descripción del funcionamiento del sistema

Dentro de las unidades de procesamiento, adecuaremos un canal de comunicación hacia un sistema experto, al cual alimentaremos con información que encontremos importante para la identificación de problemas complejos en donde la causa raíz de la falla no es explícita. A través de archivos *XML* haremos la representación de las interconexiones entre las unidades de procesamiento, así como la configuración de los comandos que se ejecutarán en cada una de éstas unidades, éste formato nos permite utilizar las distintas herramientas existentes para su edición y visualización.

La gestión de fallas se llevará a través de la administración de instrucciones o *scripts* que realizarán dos principales tipos de actividades, la primera consiste en la consulta de datos a los elementos de red y la segunda en la ejecución de acciones preventivas o correctivas. Tomando en cuenta esto, el proceso de automatización reside en la correcta selección y aplicación de estos *scripts*. Como primer paso recibimos el mensaje del evento e identificamos al elemento de la red que lo originó, luego procedemos a seleccionar de una serie de *scripts* los que nos permitan obtener un informe acerca del estado en el que se encuentra ese elemento. Ya con la información referente al estado del elemento de la red, se construye un diagnóstico sobre el estado del elemento de la red, a través de una serie de preguntas y respuestas dirigidas al elemento en cuestión y a los elementos relacionados.

Para optimizar el uso de los recursos de comunicación en la red, no es conveniente realizar todas las preguntas disponibles al elemento de la red pues toma tiempo que necesitamos para resolver otras fallas, así que nos limitamos a preguntar sólo lo que es conveniente y necesario. Para la realización de consultas, la administración de *scripts* y el flujo de un diagnóstico, las máquinas de estados son ideales, pues se definen como un conjunto de estados que representan a una situación en un momento en particular y sus relaciones a través de símbolos de entrada, haciendo que el historial de símbolos de entrada determine, para cada instante, un estado para la máquina, de forma tal que el estado final dependa únicamente de las entradas. Es decir, dependiendo del mensaje del evento y de las respuestas que obtengamos de la interacción con la red, se llevará a cabo un flujo a través de los estados que componen a la máquina. Todo esto nos será útil al tomar a un mensaje del evento como un símbolo que nos ayudará a identificar el procedimiento a seguir y a la respuesta de un *script* de consulta como el símbolo que determinará el rumbo de la transición de entre un estado y otro. Así, estos fundamentos nos serán muy útiles en el diseño de una máquina de estados que nos permita codificar la administración en la ejecución de *scripts* ante los mensajes de eventos en una red de telecomunicaciones. Podríamos pensar que es una desventaja el que una máquina de estados realice las tareas de manera secuencial, pero no es así, porque precisamente esto genera un ahorro en la comunicación hacia los elementos de la red, que solo darán respuesta a las preguntas necesarias para determinar la aplicación de los *scripts* correctivos.

Si analizamos los mensajes eventos que se presentan en las redes de telecomunicaciones, podemos observar que no todos requieren de un proceso inteligente y que una gran cantidad de ellos se ignoran, pues a veces sólo son notificaciones del funcionamiento de la red. Ante esto, podemos mejorar el desempeño del sistema en general, si sólo utilizamos al sistema experto en tareas en las que sea necesario, y para ello podemos utilizar un filtro que separe a los mensajes que no requieran de este nivel de procesamiento o correlación. Es decir, podemos procesar aquellos mensajes que no caigan en el caso de una explosión combinatoria, o en otras palabras, problemas donde la cantidad de casos y estados del sistema no sea muy amplio. La explosión combinatoria ocurre cuando queremos analizar

todos los casos posibles para poder realizar un diagnóstico. Suponiendo que cada variable de un problema tuviera solo dos estados posibles, el número de casos a cada toma de decisión crecería exponencialmente como 2^n, lo cual es muy difícil de programar y mantener de manera eficiente. Ante esto, se muestra claro, que las unidades de procesamiento que componen a los procedimientos para el procesamiento de mensajes, pueden realizar la labor de una manera adecuada, como por ejemplo en los procesos de transformación aplicados a los eventos de la red como lo son el monitorizado, filtrado y enmascaramiento de alarmas. Así como seguir el procedimiento de gestión de fallas de identificación, aislamiento, reacción, resolución y notificación.

Análisis de un sistema híbrido

La hibridación de un sistema experto y un conjunto de unidades de procesamiento es muy poderosa, porque donde el sistema experto desperdicia recursos ante problemas con poca amplitud de escenarios, ahí es donde las unidades de procesamiento desempeñan su mejor papel. Y, por otra parte, cuando las unidades de procesamiento caen en una explosión combinatoria al intentar modelar una gran amplitud de escenarios, entonces, es cuando el sistema experto utiliza mejor su potencial en la relación de hechos para generar un diagnóstico.

Un sistema experto tiene mayor potencial ante problemas que requieren de correlación entre sus variables, pero tiene problemas de rendimiento ante una gran cantidad de mensajes de eventos que no requieren correlación, para ello necesitamos de un sistema de apoyo con las siguientes características:

1. Capacidad de tomar decisiones dentro de un rango de opciones.
2. Poco uso de recursos para atender a una gran cantidad de mensajes de eventos.

Como podemos apreciar, las máquinas de estados son un sistema idóneo que nos pueden servir muy bien para realizar un manejo de mensajes de eventos, maximizando el uso de los recursos computacionales, porque pueden capturar el conocimiento para aquellas situaciones que no requieran de una elevada correlación entre las variables y así realizar el proceso de prueba de soluciones y determinar diagnósticos. Y el sistema experto es ideal para relacionar una serie de hechos que se presentan en la red para así identificar fallas de tipo raíz y fallas no explícitas.

Conclusiones

El manejo del conocimiento y su procesamiento son muy costosos en tiempo y memoria. Si pensamos en un sistema para la administración de una gran red de telecomunicaciones, en donde la cantidad de fallas es elevada, podemos tener problemas para poder responder de manera eficiente. Bajo este esquema queda claro que la mejor manera de utilizar el sistema híbrido es que las tareas que realizan los diagramas de flujo se enfoquen al tratamiento de las fallas de manera aislada y que el sistema experto se encargue de recolectar fallas e información relevante con respecto a un estado general de la red de telecomunicaciones, con la finalidad de identificar fallas raíz y fallas no explícitas.

Estrategia

Para poder construir un sistema de corrección y correlación automática de fallas que cumpla con los requerimientos enunciados en el punto anterior, hemos identificado los siguientes módulos estratégicos que conformarán al sistema. A continuación, analizaremos cada uno de estos puntos con la finalidad de construir un sistema que pueda satisfacerlos o en su caso tomar en cuenta las dificultades, las limitaciones y los riesgos que puedan aparecer durante su estudio.

Módulos para la comunicación a través de la red TCP/IP

Justificación:

Dado que los elementos de la red envían los mensajes de eventos vía *TCP/IP*, y los comandos correctivos se realizan de manera remota a través de esta misma vía, entonces necesitaremos un módulo que nos permita el envío y recepción de datos a través de este protocolo.

Requerimientos:

1. Cliente para comunicación bidireccional *TCP/IP*.
2. Servidor para comunicación bidireccional *TCP/IP*.

Estrategia:

Para que la comunicación bidireccional se lleve a cabo, necesitaremos realizar dos componentes:

1. **Servidor:** Permite a un sistema ofrecer información a varios clientes de manera simultánea.
2. **Cliente:** Permite a un sistema solicitar información a un servidor.

Decodificación de mensajes de eventos

Justificación:

Los mensajes de eventos están codificados de acuerdo a especificaciones y protocolos diferentes, así que necesitamos decodificarlos para acceder a los campos y valores que los componen.

Requerimientos:

1. Decodificadores de protocolos de mensajes de eventos.
2. Protocolo estándar para la decodificación de mensajes de eventos para el acceso a sus campos y valores.

Estrategia:

Necesitaremos tomar en cuenta para el diseño, módulos para la decodificación de los protocolos, así como un protocolo estándar para el manejo general de los mensajes de eventos por parte del sistema.

Almacenamiento de mensajes de eventos

Justificación:

Necesitamos un módulo que reciba a los mensajes de eventos y los mantenga disponibles en el mismo orden en que llegaron para su procesamiento. Hay que tomar en cuenta que la frecuencia de llegada de los mensajes no es fijo. También podría ocurrir que el procesamiento de los mensajes se vea bloqueado o interrumpido por algún motivo y esto no deberá afectar su recepción. Así que necesitaremos un contenedor que reciba una gran cantidad de mensajes en periodos cortos de tiempo, y que tenga la capacidad de contenerlos en grandes cantidades hasta su procesamiento.

Requerimientos:

1. Cola de recepción de mensajes de eventos que permita recibir más de 20 mensajes por segundo.
2. Inserción y acceso a los mensajes de manera simultánea.
3. Maximizar el uso del almacenamiento en la máquina anfitrión para la recepción de mensajes.
4. Puerto de conexión para la entrada de mensajes.

Estrategia:

Utilizaremos un contenedor *FIFO* que permita sobrecarga en el ingreso acelerado de mensajes de eventos, así como un desborde a archivos en disco duro *SWAP* sin perder el *FIFO*, de modo que pueda recargarlos en el contenedor para su procesamiento. Punto de acceso que le permita a los elementos de red enviar sus mensajes de eventos al sistema inteligente.

Identificación y categorización de mensajes de eventos

Justificación:

Los mensajes de eventos se componen de relaciones uno a uno entre campos y valores. Cada categoría es capaz de resolver una serie de fallas que inciden en un patrón determinado, este patrón se identifica dentro de los atributos del mensaje de falla para procesarlo dentro de la categoría que le corresponde.

Requerimientos:

1. Identificación de patrones en los mensajes de eventos a través de la lectura de sus campos y valores.
2. Canalizar el mensaje del evento a su categoría correspondiente.

Estrategia:

Canalizaremos al mensaje del evento dentro de unidades de procesamiento que determinarán a través de expresiones regulares, la trayectoria hacia la categoría de procesamiento correspondiente.

Representación del conocimiento para la corrección de fallas

Justificación:

La corrección de una falla se compone de un proceso iterativo de dos procesos complementarios. Uno consiste en realizar consultas a los elementos de la red relacionados para elaborar un diagnóstico, y el otro consiste en aplicar los procesos correctivos a los elementos de la red que así lo requieran. Por lo tanto, necesitamos la representación de un flujo de consultas y la aplicación de procesos consecuentes al resultado de esas consultas.

Requerimientos:

1. Diseño de un formato de archivo de fácil edición para la representación de unidades de procesamiento de consulta y de aplicación de procesos con transiciones entre ambas.
2. Manejo de excepciones y tiempos máximos de espera para los procesos.
3. Módulo de procesamiento que permita leer el archivo y genere las estructuras en memoria que permitan su aplicación.
4. La salida de la ejecución de un comando deberá poder dirigir la toma de decisiones en cuanto al la trayectoria a seguir durante procesamiento del mensaje del evento.
5. Si el procesamiento del mensaje de falla así lo requiere, la información del mensaje del evento debe poderse editar.

Estrategia:

Utilizaremos *XML* para la representación de las máquinas de estado. *XML* Son las siglas en inglés de *eXtensible Markup Language*, es decir lenguaje extensible de marcado. Es un lenguaje desarrollado por la *World Wide Web Consortium* y permite definir la gramática de lenguajes específicos. Es un estándar para el intercambio de información estructurada entre sistemas y por ello los lenguajes de programación más populares cuentan con bibliotecas su uso. Éstas características lo convierten en un formato ideal para una representación clara y flexible de los procesos en archivo.

Representación del conocimiento para la correlación de mensajes de eventos

Justificación:

Algunas fallas son consecuencia de otras, e intentar resolverlas no tiene sentido si el origen del problema no se encuentra en el dispositivo que generó el mensaje. Para determinar fallas raíz, se requiere de la correlación de datos. Como esta correlación de datos no se puede llevar a cabo de una manera eficiente a través del procesamiento de mensajes de eventos de manera independiente, es necesario contar una base de conocimiento que le permita a un proceso complementario llevar la labor de correlación.

Requerimientos:

1. Formato claro para la representación del conocimiento y diagnóstico de mensajes de eventos.

Estrategia:

Utilizaremos *CLIPS* como motor para el sistema experto. *CLIPS* Se caracteriza por ser un sistema experto rápido, eficiente y gratuito. Fue creado a partir de 1984, en el Lyndon B. Johnson Space Center de la *NASA*. *CLIPS* es un acrónimo de Sistema de Producción Integrado en Lenguaje C o *C Language Integrated Production System*. Es una herramienta muy completa que provee un entorno de desarrollo para la producción y ejecución de sistemas expertos y cuenta con las siguientes características:

1. Soporta tres paradigmas de programación: declarativo, imperativo, y orientado a objetos.
2. Se puede utilizar en sistemas con un compilador ANSI de C, o un compilador de C++.
3. Pueden escribirse extensiones a *CLIPS* sobre C, y *CLIPS* puede ser llamado desde C.
4. Incluye herramientas para la depuración y un editor integrado.
5. Contiene funcionalidades que permiten verificar las reglas incluidas en el sistema experto, así como chequeo de restricciones estático y dinámico para funciones, algunos tipos de datos y análisis semántico para prevenir inconsistencias.
6. Posee una extensa documentación.
7. Es un software de dominio público.

Procesamiento de eventos mediante el uso de unidades de procesamiento

Justificación:

Necesitamos realizar a través de un flujo de procesamiento la construcción de diagnósticos en base a preguntas y respuestas. Estas preguntas se realizarán a elementos dentro de una red compuesta de diferentes tecnologías. En base a las respuestas que nos ofrezcan las solicitudes de información a los elementos de la red, debemos poder tomar caminos correspondientes dentro del flujo del procesamiento.

Requerimientos:

1. Manejo de excepciones y tiempos máximos de espera para los procesos.
2. La salida de la ejecución de un comando deberá poder dirigir la toma de decisiones en cuanto a la trayectoria a seguir durante procesamiento del mensaje del evento.
3. Si el procesamiento del mensaje del evento así lo requiere, la información del mensaje debe poderse editar.
4. Solicitar información, realizar diagnósticos y soluciones hacia los elementos de la red.
5. Filtrar todos los mensajes de evento que lleguen de la red y que no sean de utilidad.
6. Comunicación bidireccional con distintos servicios en la red.

Estrategia:

Las unidades de procesamiento contarán con las siguientes características:

1. Existe una unidad inicial a la que se le vierten los mensajes de eventos para ser clasificadas.
2. Reciben la información del mensaje del evento y se la transmite al siguiente unidad.
3. Tienen la capacidad de leer y escribir sobre el contenido del mensaje del evento.

4. Pueden ejecutar una serie de instrucciones a través de la consola de comandos del sistema anfitrión, para así aprovechar las herramientas de comunicaciones del sistema con los elementos de la red.
5. La salida de cada unidad se toma como el siguiente símbolo de la cadena de entrada y con ello la transición a la siguiente unidad.
6. Las instrucciones que se ejecutarán en la línea de comandos del sistema operativo, podrán recibir como parámetro al mensaje del evento. Esto permitirá hacer *scripts* que realicen el filtrado de la información contenida en el mensaje, a modo que el comando que se quiera ejecutar pueda obtener esta información.

Bajo este esquema queda claro que podemos recibir al mensaje del evento en una unidad inicial y ahí mismo podemos descartarlo o procesarlo bajo los criterios que ahí se establezcan. En caso de requerir de procesamiento, se determinará a que categoría corresponde y se direccionará su flujo hacía una serie de unidades de procesamiento que servirán para atenderlo a través de una colección de comandos para realizar el diagnóstico y sus acciones correctivas.

Correlación de eventos mediante el uso de sistemas expertos

Justificación:

Los sistemas expertos son los más adecuados para realizar labores de diagnóstico. Así que necesitaremos de un proceso que realice la aplicación del conocimiento para el diagnóstico y correlación de eventos.

Requerimientos:

1. Deberá recibir información útil acerca del problemas asociados al estado de la red de telecomunicaciones, de manera que uniendo estos trozos de información, pueda realizar el diagnostico de fallas que requieren de correlación.

Estrategia:

Utilizaremos *CLIPS* por las razones descritas en el punto titulado *Representación del conocimiento para la correlación de mensajes de eventos*.

Paralelización del procesamiento de mensajes de eventos

Justificación:

Para poder agilizar el procesamiento de los mensajes de eventos utilizando los recursos del sistema de manera eficiente, podemos aprovechar el uso de hilos para su procesamiento en paralelo. Pero debemos tener cuidado en establecer un número máximo de hilos concurrentes, con la finalidad de no saturar y bloquear al sistema.

Requerimientos:

1. Gestor de hilos de procesamiento para el procesamiento de mensajes de eventos a través de las categorías correspondientes.

Estrategia:

Utilizaremos un proceso que tome a los mensajes de eventos del contenedor y cree un hilo de ejecución para su procesamiento.

Administración remota de servicios

Justificación:

Habitualmente los sistemas de gestión se encuentran en servidores bajo condiciones especiales de mantenimiento y los administradores utilizan máquinas externas para realizar las tareas dentro del servidor, es por esto que se encuentra importante poder tener la capacidad de administración a través de conexiones tipo *TCP/IP*. Necesitaremos un puerto de conexión para la gestión del sistema.

Requerimientos:

1. Módulo de recepción y respuesta a las solicitudes de información del sistema.
2. Módulo para la ejecución de comandos para la administración del sistema.

Estrategia:

En otros requerimientos ya hemos solicitado un módulo para la comunicación *TCP/IP*, así que en este requerimiento nos enfocaremos en realizar un módulo que permita la interacción de un administrador con sistema de manera remota. Esta interacción necesita de dos tipos de comandos, unos que permitan obtener información acerca del estado del sistema y otros que permitan modificar su comportamiento.

Gestor de notificaciones

Justificación:

Con la finalidad de optimizar los procesos o de realizar depuraciones de los procedimientos, necesitaremos distintos tipos de notificaciones.

Requerimientos:

1. Módulo que permita llevar la administración de las notificaciones que genere el sistema.
2. Cada componente del sistema deberá contar con el acceso a este módulo, a modo que pueda hacer la notificación con un nivel de relevancia asociado.

Estrategia:

Realizaremos el módulo que administre las notificaciones de los módulos, el criterio de esta administración se llevará a cabo a través de un nivel de relevancia que permitirá determinar si la notificación se mostrará o no.

Casos de uso

Los casos de uso proporcionan una manera incremental y modular de describir software. Definen como los usuarios utilizarán el software. Los casos de uso proporcionan una representación que puede ser fácilmente comprendida por todos los interesados. Se representan gráficamente con *Diagramas de casos de uso de UML*.

- ***Caso de uso*:** es una descripción de un conjunto de secuencias de acciones que realiza el sistema para entregar a un actor un resultado o valor observable [31]. El nombre del caso de uso debe iniciarse, preferentemente, con un verbo en infinitivo y se recomienda expresar funcionalidad en términos familiares al cliente y a los futuros usuarios.
- ***Actor*:** Es algo externo al software que intercambia información con el sistema, puede ser un usuario u otro sistema. El objetivo de un actor es completar una funcionalidad con el software para obtener un valor o servicio. Se representa con una figura humana con el nombre del actor en singular. Los sistemas externos con los que interacciona el software que se está desarrollando también son actores. Un caso de uso puede proporcionar un valor a uno o más actores.

Una vez identificados los principales casos de uso para cada actor, se describe en detalle cada uno. La plantilla que se propone para esta descripción es:

1. ***Nombre*:** El nombre deberá ser un verbo en representación de la funcionalidad del caso de uso.
2. ***Actor*:** Nombre en singular del actor encargado de la acción y que recibe el resultado del caso de uso.
3. ***Descripción*:** Texto breve describiendo la acción.
4. ***Precondiciones*:** Acciones previas del estado del sistema para que se pueda iniciar el caso de uso.
5. ***Flujo de eventos normales*:** Describe el flujo de acciones entre el actor y el sistema.
6. ***Flujo de eventos alternativos o excepcionales*:** Acciones que ocurren en situaciones anormales.
7. ***Poscondiciones*:** Define el estado del sistema después de la terminación exitosa del caso de uso.

A continuación se presentan los casos de uso que considero pertinentes para el desarrollo del sistema inteligente.

Lista de casos de uso:

1. Encender el sistema.
2. Mostrar el estado general de los procesos del sistema.
3. Mostrar el estado del contenedor.
4. Insertar un mensaje de evento dentro del contenedor.
5. Mostrar el estado del servidor de gestión.

6. Obtener el nombre del servidor de gestión.
7. Mostrar el estado del servidor de recepción de mensajes de eventos.
8. Obtener el nombre del servidor de recepción de mensajes de eventos.
9. Mostrar el estado del gestor de procesamiento de mensajes de eventos.
10. Iniciar o reiniciar el procesamiento de mensajes de eventos.
11. Pausar el procesamiento de mensajes de eventos.
12. Mostrar los elementos que componen a las unidades de procesamiento.
13. Cargar un archivo que describe a una máquina de estados.
14. Establecer el nivel de detalle de los mensajes del sistema.
15. Establecer si se mostrará la ruta de origen completa de los mensajes del sistema.
16. Apagar el sistema.
17. Encender el sistema experto.
18. Cargar un archivo con sentencias en el sistema experto.
19. Interactuar con el sistema experto.
20. Apagar el sistema experto.

Encender el sistema

Descripción:	Arranca el sistema incluyendo a los puertos de acceso.
Precondiciones:	El archivo de configuraciones debe estar correctamente constituido, así como los recursos necesarios disponibles, en especial los puertos y la memoria suficiente.
Poscondiciones:	El sistema se encuentra escuchando en los puertos en espera de mensajes de eventos o instrucciones.

Flujo de eventos normales:

	USUARIO		**SISTEMA**	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Introduce el comando relacionado al inicio del sistema, a través de la terminal y lo ejecuta.	2	Inicializa todos los módulos y servicios.	**E1, E2, E3**

Flujo de eventos alternativos:

ID	**NOMBRE**	**ACCIÓN**
E1	El archivo de configuraciones no esta correctamente constituido.	Dependiendo del nivel de error, se cierra el programa o se inicia con valores por defecto.
E2	Algún puerto no se encuentra disponible.	Se cierra el programa.
E3	Memoria insuficiente.	Se cierra el programa.

Mostrar el estado general de los procesos del sistema

Descripción:	Muestra el estado general del sistema a través de una conexión por *socket*.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para

	realizar la solicitud.
Poscondiciones:	La información acerca del estado general del sistema se ha enviado a través del *socket* que la ha solicitado.

Flujo de eventos normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud de información del estado general del sistema.	2	Envía el estado general del sistema en respuesta a la solicitud del *socket*.	**E1**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.

Mostrar el estado del contenedor

Descripción:	Muestra el estado del contenedor a través de una conexión por *socket*.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud.
Poscondiciones:	La información acerca del estado del contenedor se ha enviado a través del *socket* que la ha solicitado.

Flujo de eventos normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud de información del contenedor.	2	Envía el estado del contenedor en respuesta a la solicitud del *socket*.	**E1**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.

Insertar un mensaje de evento dentro del contenedor

Descripción:	Inserta un mensaje de evento de manera manual dentro del contenedor a través de

	una conexión por *socket*.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la inserción.
Poscondiciones:	El contenedor ahora cuenta con un mensaje de evento que ha sido insertado de manera manual.

Flujo de eventos normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Escribe el comando que corresponde a la inserción de un mensaje de evento en el contenedor.	2	Inserta un mensaje de evento en el contenedor y avisa al usuario que el proceso de inserción se ha llevado acabo de forma satisfactoria.	**E1, E2**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.
E2	Memoria insuficiente.	El sistema envía un mensaje de error.

Mostrar el estado del servidor de gestión

Descripción:	Muestra el estado del servidor de gestión a través de una conexión por *socket*.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud.
Poscondiciones:	La información acerca del servidor de gestión se ha enviado a través del *socket* que la ha solicitado.

Flujo de eventos normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud de información del servidor de gestión.	2	Envía el estado del servidor de gestión en respuesta a la solicitud del *socket*.	**E1**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.

Obtener el nombre del servidor de gestión

Descripción:	Muestra el nombre del servidor de gestión a través de una conexión por *socket*.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud.
Poscondiciones:	El nombre del servidor de gestión se ha enviado a través del *socket* que lo ha solicitado.

Flujo de eventos normales:

	USUARIO		**SISTEMA**	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud del nombre del servidor de gestión.	2	Envía el nombre del servidor de gestión en respuesta a la solicitud del *socket*.	**E1**

Flujo de eventos alternativos:

ID	**NOMBRE**	**ACCIÓN**
E1	Conexión interrumpida.	El sistema envía un mensaje de error.

Mostrar el estado del servidor de recepción de mensajes de eventos

Descripción:	Muestra el estado del servidor de recepción de mensajes de eventos a través de una conexión por *socket*.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud.
Poscondiciones:	La información acerca del servidor de recepción de mensajes de eventos se ha enviado a través del *socket* que la ha solicitado.

Flujo de eventos normales:

	USUARIO		**SISTEMA**	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud de información del servidor de recepción de mensajes de eventos.	2	Envía el estado del servidor de recepción de mensajes de eventos en respuesta a la solicitud del *socket*.	**E1**

Flujo de eventos alternativos:

ID	**NOMBRE**	**ACCIÓN**
E1	Conexión interrumpida.	El sistema envía un mensaje de error.

Obtener el nombre del servidor de recepción de mensajes de eventos

Descripción:	Muestra el nombre del servidor de recepción de mensajes de eventos a través de una conexión por *socket*.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud.
Poscondiciones:	El nombre del servidor de recepción de mensajes de eventos se ha enviado a través del *socket* que lo ha solicitado.

Flujo de eventos normales:

	USUARIO		**SISTEMA**	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud del nombre del servidor de recepción de mensajes de eventos.	2	Envía el nombre del servidor de recepción de mensajes de eventos en respuesta a la solicitud del *socket*.	**E1**

Flujo de eventos alternativos:

ID	**NOMBRE**	**ACCIÓN**
E1	Conexión interrumpida.	El sistema envía un mensaje de error.

Mostrar el estado del gestor de procesamiento de mensajes de eventos

Descripción:	Muestra el estado del gestor de procesamiento de mensajes de eventos a través de una conexión por *socket*.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud.
Poscondiciones:	La información acerca del gestor de procesamiento de mensajes de eventos se ha enviado a través del *socket* que la ha solicitado.

Flujo de eventos normales:

	USUARIO		**SISTEMA**	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud de información del gestor de procesamiento de mensajes de eventos.	2	Envía el estado del gestor de procesamiento de mensajes de eventos en respuesta a la solicitud del *socket*.	**E1**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.

Iniciar o reiniciar el procesamiento de mensajes de eventos

Descripción:	Inicia o reanuda al procesador de mensajes de eventos.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud. El procesador de mensajes de eventos se debe encontrar en estado de no iniciado o en pausa.
Poscondiciones:	El procesamiento de mensajes de eventos se encuentra activo.

Flujo de fallas normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud de inicio o reinicio del gestor de procesamiento de mensajes de eventos.	2	Inicia o reanuda el procesamiento de mensajes de eventos y envía el estado del gestor de procesamiento de eventos en respuesta a la solicitud del *socket*.	**E1**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.

Pausar el procesamiento de mensajes de eventos

Descripción:	Una vez iniciado el procesador de mensajes de eventos, no se puede apagar de manera independiente al sistema, sólo se puede pausar.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud. El procesador de mensajes de eventos se debe encontrar activo.
Poscondiciones:	El procesador de mensajes de eventos se encuentra en pausa.

Flujo de eventos normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en	**2**	Pausa el procesamiento de los	**E1**

	un *shell*, la solicitud de pausa al gestor de procesamiento de mensajes de eventos.		mensajes de eventos y envía el estado del gestor de procesamiento de mensajes de eventos en respuesta a la solicitud del *socket*.	

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.

Mostrar los elementos que componen a las unidades de procesamiento

Descripción:	Muestra la lista de componentes, sus transiciones y comandos de ejecución de la máquina de estados.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud.
Poscondiciones:	La información acerca de los componentes de la máquina de estados se ha enviado a través del *socket* que la ha solicitado.

Flujo de eventos normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud de información de la máquina de estados.	2	Envía la información acerca de los componentes de la máquina de estados en respuesta a la solicitud del *socket*.	**E1**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.

Cargar un archivo que describe a una máquina de estados

Descripción:	A partir de un archivo, se pueden cargar estados y transiciones que sirvan para construir a la máquina de estados que atenderá a los mensajes de eventos.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud.
Poscondiciones:	Los estados y las transiciones definidos en el archivo se han cargado dentro de la máquina de estados del sistema.

Flujo de eventos normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud de carga del archivo.	2	El archivo se carga dentro de la máquina de estados del sistema y envía el resultado del proceso de carga del archivo en respuesta a la solicitud del *socket*.	**E1, E2**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.
E2	Error al cargar el archivo.	El sistema envía un mensaje con la descripción del error que ha ocurrido en la carga del archivo.

Establecer el nivel de detalle de los mensajes del sistema

Descripción:	El sistema cuenta con un gestor de mensajes del sistema que puede ser configurando por niveles de detalle de exposición.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud.
Poscondiciones:	El nivel de detalle de los mensajes del sistema se ha actualizado al valor proporcionado.

Flujo de eventos normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud de actualización del valor del nivel de detalle en los mensajes del sistema.	2	Se actualiza el valor y envía el resultado del proceso en respuesta a la solicitud del *socket*.	**E1**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.

Establecer si se mostrará la ruta de origen completa de los mensajes del sistema

Descripción:	El sistema cuenta con un gestor de mensajes del sistema que puede ser

	configurando para mostrar o no, la ruta completa del origen del mensaje.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud.
Poscondiciones:	Se ha establecido si se mostrará o no, la ruta completa del origen del mensaje.

Flujo de eventos normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud de actualización del valor que define si se mostrará o no la ruta completa del origen del mensaje.	2	Se actualiza el valor y envía el resultado del proceso en respuesta a la solicitud del *socket*.	**E1**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.

Apagar el sistema

Descripción:	El sistema requiere de ser apagado de manera ordenada, para permitir el cierre adecuado de los distintos módulos que lo conforman.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud.
Poscondiciones:	El sistema ha terminado su ejecución y terminado a cada uno de sus módulos de manera adecuada.

Flujo de eventos normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud de apagado del sistema.	2	Inicia el apagado del sistema, se manda la señal a cada uno de los módulos y envía el resultado del proceso en respuesta a la solicitud del *socket*.	**E1**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.

Encender el sistema experto

Descripción:	Arranca el sistema experto.
Precondiciones:	En el sistema anfitrión, se debe encontrar el puerto de comunicación disponible y la memoria suficiente.
Poscondiciones:	El sistema experto se encuentra escuchando por el puerto asignado.

Flujo de eventos normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Introduce el comando relacionado al inicio del sistema, a través de la terminal y lo ejecuta.	2	Inicializa todos los módulos y servicios.	**E1, E2, E3**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Se proporcionaron de manera incorrecta los parámetros para iniciar el sistema.	Dependiendo del nivel de error, se cierra el programa o se inicia con valores por defecto.
E2	El puerto no se encuentra disponible.	Se cierra el programa.
E3	Memoria insuficiente.	Se cierra el programa.

Cargar un archivo con sentencias en el sistema experto

Descripción:	A partir de un archivo, se pueden cargar sentencias para constituir la base de conocimientos del sistema experto.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud.
Poscondiciones:	Las sentencias definidas en el archivo se han cargado dentro de la base de conocimientos del sistema experto.

Flujo de eventos normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud de carga del archivo.	2	El archivo se carga dentro de la base de conocimientos del sistema experto y envía el resultado del proceso de carga del archivo en respuesta a la solicitud del *socket*.	**E1, E2**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.
E2	Error al cargar el archivo.	El sistema envía un mensaje con la descripción del error que ha ocurrido en la carga del archivo.

Interactuar con el sistema experto

Descripción:	Poder enviar a través de un *socket* comandos al sistema experto y recibir respuesta.
Precondiciones:	El sistema debe estar encendido y establecida una conexión por *socket* para realizar la solicitud.
Poscondiciones:	Los comandos han sido procesados y se les ha dado respuesta.

Flujo de eventos normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket*, los comandos al sistema experto.	2	El sistema experto procesa los comandos y envía la respuesta del sistema experto al usuario.	**E1, E2**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.
E2	Error en la sintaxis del comando.	El sistema envía un mensaje con la descripción del error que ha ocurrido al intentar ejecutar el comando.

Apagar el sistema experto

Descripción:	El sistema experto requiere de ser apagado.
Precondiciones:	El sistema experto debe estar encendido y establecida una conexión por *socket* para realizar la solicitud.
Poscondiciones:	El sistema ha terminado su ejecución.

Flujo de eventos normales:

	USUARIO		SISTEMA	
PASO	**ACCIÓN**	**PASO**	**ACCIÓN**	**EXCEPCIONES**
1	Envía a través de un *socket* en un *shell*, la solicitud de apagado del sistema experto.	2	Inicia el apagado del sistema experto y envía el resultado del proceso en respuesta a la solicitud del socket.	**E1**

Flujo de eventos alternativos:

ID	NOMBRE	ACCIÓN
E1	Conexión interrumpida.	El sistema envía un mensaje de error.

Componentes

A continuación se hace una descripción de los componentes del sistema y más adelante una relación de estos componentes con los requerimientos identificados.

Servidor de CLIPS

Permite interactuar con una instancia de *CLIPS* a través de *sockets* para realizar labores de diagnóstico, relacionadas con la correlación de fallas.

Requerimientos que cumple:

- Recibe información útil acerca del problemas asociados al estado de la red de telecomunicaciones, de manera que al unir estos trozos de información, puede realizar el diagnostico de fallas que requieren de correlación.
- Utiliza un formato claro para la representación del conocimiento, diagnóstico y la generación de fallas producto de la correlación.

Módulo lector de XML

Permite leer archivos *XML* en los cuales se encuentran codificados los procedimientos de identificación y corrección de fallas mediante la administración de *scripts*.

Requerimientos que cumple:

- Utilizaremos *XML* para la representación de las máquinas de estado, por ser un formato ideal para una representación clara y flexible de los procesos en archivo.
- Manejo de excepciones y tiempos máximos de espera para los procesos.
- Utiliza un módulo de procesamiento que permite leer el archivo *XML* y genera a partir de él las

estructuras en memoria que permite la aplicación del proceso codificado.

- Cada estado puede realizar la ejecución de un comando instalado en el sistema operativo anfitrión para realizar la comunicación con los elementos de la red, cuya salida dirige la toma de decisiones en cuanto a la trayectoria a seguir durante procesamiento del mensaje de evento.
- Si el procesamiento del mensaje de evento así lo requiere, la información del mensaje debe poderse editar.
- El procesamiento de un mensaje de evento, se compone de un proceso iterativo de dos procesos complementarios. Uno consiste en realizar consultas a los elementos de la red relacionados para elaborar un diagnóstico. Y el otro consiste en aplicar los procesos correctivos a los elementos de la red que así lo requieran.
- Realiza la representación de un flujo de consultas y la aplicación de procesos consecuentes al resultado de esas consultas.

Módulo de comunicaciones

Dado que los elementos de la red envían los reportes utilizando el protocolo estándar *TCP/IP*, y los comandos correctivos se realizan de manera remota a través de esta misma vía, entonces son necesarios módulos que permitan el envío y recepción de datos a través de este protocolo. Cada instancia permite realizar las comunicaciones necesarias entre los componentes a través de la red.

Requerimientos que cumple:

- Para llevar a cabo la comunicación bidireccional, se desarrollaron dos componentes:
  - **Módulo Servidor:** Permite a un sistema ofrecer información a varios clientes de manera simultánea.
  - **Módulo Cliente:** Permite a un sistema solicitar información a un servidor.
- Las fallas están codificadas de acuerdo a especificaciones y protocolos diferentes, pero en el alcance del presente proyecto nos limitamos a recibir las fallas en código *ASCII* y a través del protocolo *TCP/IP*.

Unidades de procesamiento

Permiten la representación de los procedimientos al relacionarlos entre sí. Cada unidad tiene asociado un *script* que será ejecutado al llegar a ella. Necesitamos realizar a través de un flujo de procesamiento la construcción de diagnósticos en base a preguntas y respuestas. Estas preguntas se realizan a elementos dentro de una red compuesta de diferentes tecnologías. En base a las respuestas que nos ofrecen las solicitudes de información a los elementos de la red, podemos tomar caminos correspondientes de entre un conjunto de unidades de procesamiento para el procesamiento de los mensajes de eventos. Para agilizar el procesamiento de los mensajes utilizamos hilos de ejecución para realizar a los procesos de manera simultánea.

Requerimientos que cumple:

- Existe una unidad inicial a la que se le vierten los mensajes de eventos para ser clasificados.

- Se canalizan los mensajes de eventos dentro de unidades de procesamiento que determinan la trayectoria hacia la categoría de procesamiento correspondiente.
- Se pueden filtrar todos los mensajes de eventos que llegan de la red y que no son de utilidad.
- Identifica patrones en los mensajes de eventos a través de la lectura de sus campos y valores.
- Maneja excepciones y tiempos máximos de espera para los procesos.
- Se puede solicitar información, realizar diagnósticos y aplicar soluciones hacia los elementos de la red.
- La información del mensaje del evento se transmite de unidad en unidad.
- Capacidad de modificar la información del mensaje del evento a su paso por las unidades de procesamiento.
- Las unidades de procesamiento pueden ejecutar los comandos del sistema operativo, y así se pueden utilizar las herramientas de comunicación hacia los elementos de la red. La salida de cada unidad se toma como el siguiente símbolo de la cadena de entrada y con ello la transición a la siguiente unidad.
- Los comandos que se ejecutarán en el sistema operativo cada unidad de procesamiento, podrán recibir como parámetro a un mensaje de falla. Esto permite hacer *scripts* que realicen el filtrado de la información contenida en el mensaje del evento, a modo que el comando que se quiere ejecutar puede obtener esta información de manera adecuada.
- Se puede establecer un número máximo de hilos concurrentes, para no saturar y bloquear al sistema.
- Existe un proceso que toma a los mensajes de eventos del contenedor y crea un hilo de ejecución para su procesamiento.

Contenedor de mensajes de eventos

Permite la recepción de mensajes de eventos en memoria, para que al saturarse haga desbordamientos a disco duro y conforme se van procesado los mensajes, se cargan los bloques del disco duro a memoria nuevamente, sin perder el orden.

Requerimientos que cumple:

- La recepción de hasta 300 mensajes de eventos por segundo.
- Si el procesamiento de los mensajes de eventos se ve bloqueado o interrumpido por algún motivo, su recepción continúa aunque lleguen en grandes cantidades, mientras el disco duro lo pueda almacenar hasta su procesamiento.
- Se pueden realizar inserciones y lecturas de mensajes de eventos de manera simultánea.
- Utiliza un contenedor *FIFO* que permite un desborde a archivos en disco duro *SWAP* sin perder el *FIFO*, de modo que puede recargarlos en el contenedor para su procesamiento.

Shell

Es una interfaz de usuario para nuestro sistema y el servidor de *CLIPS*, nos permite conocer el estado del sistema y el servidor, así como para establecer configuraciones y llevar a cabo su administración a

través de conexiones tipo *TCP/IP*.

Requerimientos que cumple:

- Módulo de recepción y respuesta a las solicitudes de información del sistema.
- Módulo para la ejecución de comandos para la administración del sistema.
- Permite la interacción de un administrador con el sistema de manera remota mediante dos grupos de comandos, unos que permiten obtener información acerca del estado del sistema y otros que permiten modificar su comportamiento.

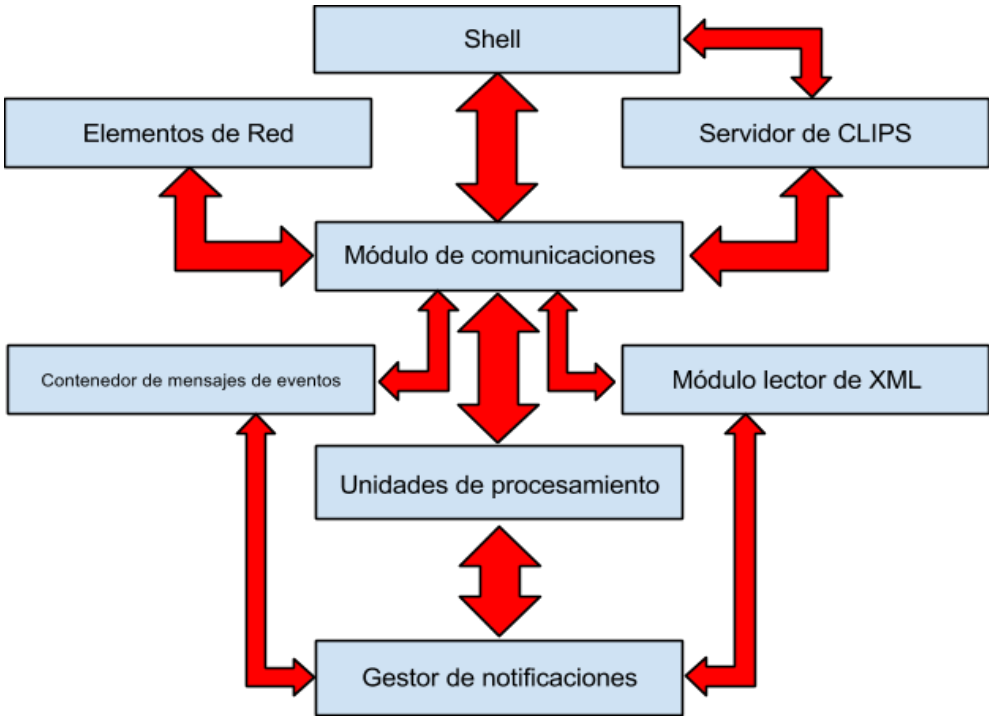
Gestor de Notificaciones

Permite a los componentes del sistema, contar con una interfaz para el envío de mensajes a los administradores y usuarios del sistema con la finalidad de optimizar procesos o realizar depuraciones de los procedimientos.

Requerimientos que cumple:

- Permite llevar la administración de las notificaciones que genere el sistema.
- Cada componente del sistema cuenta con acceso a este módulo.
- Las notificaciones cuentan con un nivel de relevancia que permite al usuario determinar el nivel de profundidad y de detalle de las notificaciones del sistema.

Diagrama de interacciones



Descripción

A través del *shell* se puede realizar el siguiente procedimiento:

- Se realizan las configuraciones correspondientes.
- Se carga el archivo de procesos que será representado en las unidades de procesamiento.
- Se cargarán las reglas y los hechos en el servidor de *CLIPS*.
- Se inicia el servicio de procesamiento de mensajes de eventos.
- Se realiza la monitorización de los procesos a través del gestor de notificaciones.
- Se detiene el procesamiento de mensajes de eventos.
- Se detiene el servidor de *CLIPS*.
- Se detiene el sistema.

El flujo habitual del mensaje de evento podría configurarse como sigue:

- Se genera el mensaje de evento por algún elemento de red.
- El mensaje de evento puede pasar por otros elementos de red hasta llegar al sistema por el módulo de comunicaciones.
- Se inserta el mensaje de evento en la cola del contenedor.
- Las unidades de procesamiento toman un mensaje de evento del contenedor y lo procesan generando interacciones con los elementos de red involucrados.
- Termina el proceso.

Tabla de relación entre componentes y requerimientos

A continuación se muestra una relación de los componentes con los requerimientos, la cual es plena.

MÓDULOS	GESTOR DE NOTIFICACIONES	SHELL	CONTENEDOR DE EVENTOS	U. DE PROCESAMIENTO	MÓD. DE COMUNICACIONES	MÓD. LECTOR DE XML	SERVIDOR DE CLIPS
CASOS DE USO							
ENCENDER EL SISTEMA	•	•	•		•		
MOSTRAR EL ESTADO GENERAL DE LOS PROCESOS DEL SISTEMA	•	•	•	•	•		
MOSTRAR EL ESTADO DEL CONTENEDOR	•	•	•		•		
INSERTAR UN MENSAJE DE EVENTO DENTRO DEL CONTENEDOR	•	•			•		
MOSTRAR EL ESTADO DEL SERVIDOR DE GESTIÓN	•	•			•		
OBTENER EL NOMBRE DEL SERVIDOR DE GESTIÓN	•	•			•		
MOSTRAR EL ESTADO DE SERVIDOR DE RECEPCIÓN DE MENSAJES DE EVENTOS	•	•			•		
OBTENER EL NOMBRE DE SERVIDOR DE RECEPCIÓN DE MENSAJES DE EVENTOS	•	•			•		
MOSTRAR EL ESTADO DEL GESTOR DE PROCESAMIENTO DE MENSAJES DE EVENTOS	•	•		•	•		
INICIAR O REINICIAR EL PROCESAMIENTO DE MENSAJES DE EVENTOS	•	•		•	•		
PAUSAR EL PROCESAMIENTO DE MENSAJES DE EVENTOS	•	•		•	•		
MOSTRAR LOS ELEMENTOS QUE COMPONEN A LAS UNIDADES DE PROCESAMIENTO	•	•		•	•		
CARGAR UN ARCHIVO QUE DESCRIBE A UNA MÁQUINA DE ESTADOS	•	•		•	•	•	
ESTABLECER EL NIVEL DE DETALLE DE LOS MENSAJES DEL SISTEMA	•	•			•		
ESTABLECER SI SE MOSTRARÁ LA RUTA DE ORIGEN COMPLETA DE LOS MENSAJES DEL SISTEMA	•	•			•		
APAGAR EL SISTEMA	•	•		•	•		
ENCENDER EL SISTEMA EXPERTO	•	•					•
CARGAR UN ARCHIVO CON SENTENCIAS EN EL SISTEMA EXPERTO	•	•					•
INTERACTUAR CON EL SISTEMA EXPERTO	•	•					•
APAGAR EL SISTEMA EXPERTO	•	•					•
MÓDULOS IDENTIFICADOS EN LOS REQUERIMIENTOS							
MÓDULOS PARA LA COMUNICACIÓN A TRAVÉS DE LA RED *TCP/IP*		•			•		•
DECODIFICACIÓN DE MENSAJES DE EVENTOS		•	•		•		
ALMACENAMIENTO DE MENSAJES DE EVENTOS			•				
IDENTIFICACIÓN Y CATEGORIZACIÓN DE MENSAJES DE EVENTOS				•			•
REPRESENTACIÓN DEL CONOCIMIENTO PARA LA CORRECCIÓN DE FALLAS				•		•	
REPRESENTACIÓN DEL CONOCIMIENTO PARA LA CORRELACIÓN DE EVENTOS							•
PROCESAMIENTO DE EVENTOS MEDIANTE EL USO DE UNIDADES DE PROCESAMIENTO				•			
CORRELACIÓN DE EVENTOS MEDIANTE EL USO DE SISTEMAS EXPERTOS							•
PARALELIZACIÓN DEL PROCESAMIENTO DE MENSAJES DE EVENTOS				•			
ADMINISTRACIÓN REMOTA DE SERVICIOS	•	•			•	•	
GESTOR DE NOTIFICACIONES	•	•		•	•	•	
OTROS REQUERIMIENTOS							
RECEPCIÓN DE MENSAJES DE EVENTOS		•	•		•		
ADMINISTRACIÓN REMOTA	•	•			•	•	
PROCESAMIENTO DE MENSAJES DE EVENTOS EN PARALELO				•			
UNIDADES DE PROCESAMIENTO				•		•	

Análisis de la propuesta

Alcances

En el desarrollo de este proyecto se logra contar con un sistema capaz de ejecutar procedimientos de manera automática para la correlación de eventos y la corrección de fallas, a través de la toma de decisiones y la comunicación con los elementos de red.

Para ello utilizamos una arquitectura que resuelve el problema antes descrito, en donde la recepción de mensajes de eventos se realiza a través del desarrollo de un contenedor *FIFO* que permite su desborde a disco duro, el procesamiento en paralelo se logra mediante el desarrollo de un gestor de hilos para el procesamiento de los mensajes de eventos y la generación de diagnósticos para la correlación mediante una arquitectura con dos procesos simultáneos: el primer proceso atiende a los mensajes de eventos de manera local con máquinas de estados que recolectan información que podría ser útil para los diagnósticos complejos, y el segundo proceso analiza toda la información generada por el primer proceso con un sistema experto para determinar si existe alguna falla de correlación.

Esta arquitectura cumple con facilidades para ser:

- **Escalable:** Pues se encuentra conformada por módulos de fácil integración.
- **Paralelizable:** Para aumentar el rendimiento se puede pensar en distribuir instancias del sistema a través de distintos equipos y que cada conjunto de equipos cuente con un servicio de sistema experto para la correlación de eventos por zonas.
- **Flexible:** Porque se puede modificar su arquitectura con facilidad para adaptarse a nuevas soluciones que mejoren el desempeño actual, pues cada módulo tiene bien definidas sus tareas lo que permite desacoplarlos para integrar nuevos componentes con facilidad.

Limitaciones

Si bien bajo el esquema de una gran red de telecomunicaciones de proveedores mixta no es factible desarrollar todos los procesos correctivos para todas las fallas posibles, tener procesos para las fallas más comunes reduce la cantidad de trabajo significativamente y eleva la calidad del servicio al reducir los tiempos de respuesta.

En el desarrollo del sistema no abordaremos las dificultades que existen para integrarlo dentro de un gestor de red, pues fundamentalmente sólo nos centramos en el diseño de una arquitectura que nos permite satisfacer las necesidades del problema descrito, ocupándonos de la gestión automática de fallas a través de diversos métodos computacionales, de entre ellos el uso de la inteligencia artificial para la corrección de fallas y correlación de eventos.

Es importante aclarar, que este sistemas de inteligencia artificial requiere del rol de un experto humano, que se encargue de codificar el conocimiento de los técnicos encargados de la administración de la red, pues el sistema no es capaz de encontrar nuevos patrones para la correlación de eventos, pero si cuenta con las facilidades para integrar los procedimientos de identificación y corrección de fallas al sistema.

Se puede pensar en utilizar técnicas de minería de datos para la identificación de causas raíz con la finalidad de generar los procedimientos de identificación de causas, pero los resultados de estos análisis por ahora deben estar sujetos al análisis del experto humano antes de integrarse al sistema en producción.

Estrategia de un sistema híbrido

La propuesta que presento en esta tesina, está basada en la generación de diagnósticos dentro de una arquitectura de dos procesos simultáneos:

- **Procesamiento local**: Este proceso atiende a los mensajes de eventos con unidades de procesamiento que corrigen los problemas de manera local y aislada, y durante su ejecución van recolectando información útil para diagnósticos de fallas no explícitas, o para encontrar fallas que son causa raíz de otras fallas.
- **Procesamiento global**: Aquí es donde el sistema analiza con un sistema experto la información generada en el primer proceso y determina si existe una falla no explícita.

Procesar los mensajes de eventos de ésta manera permite distribuir la carga de trabajo de manera eficiente, pues el sistema experto consume muchos recursos a cada inserción de información, y así a través de la programación de unidades de procesamiento, se puede tratar a la mayoría de mensajes de eventos de manera económica y sólo insertamos la información necesaria al sistema experto para la correlación, minimizando redundancias para realizar mejores diagnósticos.

La hibridación de un sistema experto y un conjunto de unidades de procesamiento es muy poderosa, porque donde el sistema experto desperdicia recursos ante problemas con poca amplitud de escenarios, ahí es donde las unidades de procesamiento desempeñan su mejor papel. Y, por otra parte, cuando las unidades de procesamiento caen en una explosión combinatoria al intentar modelar una gran amplitud de escenarios, entonces, es cuando el sistema experto utiliza mejor su potencial en la relación de hechos para generar un diagnóstico.

Las condiciones ideales en las cuales funciona mejor esta arquitectura, se dan cuando la relación del tiempo que requieren los mensajes de eventos para ser procesados y la llegada de mensajes que requieren atención pronta se encuentra equilibrada. Pues podría encontrarse el contenedor de mensajes de eventos con un número muy grande de mensajes que tendrán que esperar largo tiempo para su procesamiento. También es importante recordar que este sistema está pensado para encontrarse dentro de un *OSS* o un *MOM*, pero las tareas de integración dentro de estos sistemas queda fuera de los alcances de esta tesina. Por otro lado estamos partiendo de la idea de que se cuenta con el rol de un experto humano que se encargará de codificar el conocimiento de los técnicos encargados de la administración de la red, para así integrar los procedimientos de identificación y corrección al sistema.

Antes que nada debe quedar claro que este es un sistema general, que sirve para la identificación y corrección de problemas, y necesita ser programado para resolver problemas específicos bajo el paradigma híbrido que hemos mencionado, en donde las máquinas de estado reciben el grueso de los mensajes de eventos y el sistema experto sólo es alimentado con información relevante para realizar la correlación. La manera en que se puede sacar el máximo potencial a este sistema, es retornando a la

serie de características que nos llevaron a él. Recordemos que el propósito de las unidades de procesamiento es recibir a los mensajes de eventos en su unidad de procesamiento inicial para realizar la mayor cantidad de filtrado posible, y lo que se busca durante el recorrido de los estados, es ir construyendo un diagnóstico de la falla, a través de una serie de preguntas y respuestas que se realizan dentro del conjunto de instrucciones que cada estado posee. De esta manera podemos enfocar la tarea del sistema experto en relacionar los eventos para determinar diagnósticos a nivel general, y podemos mantener actualizado su conocimiento sobre el estado de los elementos de la red, si las unidades de procesamiento brindan información relevante acerca de una falla cuando ésta pudiera estar relacionada de algún modo con otras fallas.

Con un diagnóstico adecuado el sistema puede decidir entre solucionar la falla, reducir su impacto o simplemente ignorarlo. Cuando un elemento de red envía un mensaje de evento, este mensaje debe llegar al sistema experto en caso de ser útil en la identificación de fallas que se propagan por la red. Para el manejo de fallas en una red, necesitamos tener procedimientos para realizar un diagnóstico y procedimientos para efectuar la corrección o dar aviso de una falla que no pudo ser corregida.

Codificación de los procedimientos para la identificación y corrección de fallas

Cuando un tipo de falla es frecuente, se puede encontrar un patrón que nos permita identificarlas y un procedimiento que nos permita resolverlas. Estos patrones y estos procedimientos pueden ser codificados de manera que puedan ejecutarse de manera automática. Para que esto ocurra necesitamos un sistema que nos permita recibir las fallas y darles un tratamiento clasificándolas dentro de una serie de patrones que corresponden a una serie de procedimientos que se encargarán de resolverlas. En caso de que la falla no pueda ser identificada por algún patrón o no pueda ser resuelta por ningún procedimiento, entonces se hará la notificación correspondiente al centro de servicio encargado de la administración de la red, para atenderla de manera personalizada. Todo esto lo haremos partiendo del hecho de que existen técnicos que se ocupan del mantenimiento de la red y que han identificado ya a estos patrones y a sus respectivos procedimientos. Como trabajo a futuro quedará el estudiar métodos de minería de datos que permitirán identificar de manera automática a los patrones para la categorización y posiblemente se podría pensar en la generación automática de procedimientos correctivos. El diseño de esta propuesta es útil porque permite programar y establecer procedimientos para realizar la corrección de las fallas de manera automática. Como lo son la recepción masiva de mensajes de eventos, su procesamiento en paralelo y relacionar una serie de síntomas inconexos para realizar el diagnóstico de una falla cuya falla raíz no es evidente. Al realizarse las correcciones de manera automática, se reducen los tiempos de espera para su procesamiento, lo cual permite reducir los tiempos de falla de manera considerable. Así mismo, esta propuesta permite codificar procedimientos que permitan la correlación de eventos y la identificación de fallas raíz.

Configuraciones para pruebas al sistema

Se cuenta con dos tipos de pruebas, una es la de correlación y la otra es una serie de pruebas generales. Realizamos diversas pruebas al sistema para corroborar la utilidad del diseño propuesto. Por un lado realizamos la programación de distintos procesos que permiten la validación del funcionamiento correcto del sistema en la ejecución de las mismas, y por otro lado realizamos pruebas de estrés al establecer a través de máquinas virtuales múltiples conexiones al sistema, para provocar así la

saturación del contenedor de fallas y su desbordamiento a disco duro, así como la saturación de hilos de ejecución simultáneos, esto con la finalidad de conocer las capacidades del sistema para la gestión automática de fallas.

Configuración del sistema para las pruebas.

Nuestro sistema lee un archivo de configuraciones al iniciar, que le asigna ciertas características de funcionamiento, a continuación se encuentra el contenido del archivo de configuraciones que se describirá más adelante:

Contenido del archivo de configuraciones

```
01 NetworkEventManagerHostName=NetworkEventManager_1
02
03 BufferSize=1000
04 BufferBlockSize=500
05 BufferSwapPath=swap
06
07 AllocatorWaitEmptyBuffer=1000
08 AllocatorWaitThreadsBusy=1000
09 AllocatorWaitPaused=1000
10 AllocatorThreads=100
11
12 ServerServiceHostName=NetworkEventManagerServices_1
13 ServerServicePort=8001
14 ServerServiceMaximumConnections=100
15 ServerServiceTimeout=300000
16
17 ServerHttpHostName=NetworkEventManagerHttp_1
18 ServerHttpPort=8002
19 ServerHttpMaximumConnections=100
20 ServerHttpTimeout=300000
21
22 ServerEventHostName=NetworkEventManagerEvents_1
23 ServerEventPort=8003
24 ServerEventMaximumConnections=100
25 ServerEventTimeout=300000
```

Descripción de los campos de las configuraciones:

Nombre del Parámetro	Descripción
NetworkEventManagerHostName	Es el nombre que recibirá esa instancia del sistema.
BufferSize	Tamaño del contenedor de mensajes de eventos en número de cadenas de texto.
BufferBlockSize	Tamaño de bloque para desborde a disco duro.
BufferSwapPath	Dirección en el disco duro donde se guardarán los desbordamientos.
AllocatorWaitEmptyBuffer	El tiempo que esperará el administrador de tareas para leer un nuevo mensaje de evento cuando el contenedor se encuentre vacío.
AllocatorWaitThreadsBusy	El tiempo que esperará el administrador de tareas para volver a intentar levantar un proceso, cuando la cantidad máxima de procesos haya sido alcanzada.

AllocatorWaitPaused	El tiempo que esperará el administrador de tareas para volver a intentar continuar con sus tareas cuando ha sido detenido.
AllocatorThreads	Cantidad máxima de procesos concurrentes, uno por mensaje de evento.
ServerServiceHostName	Nombre del servidor para la interfaz de usuario vía terminal.
ServerServicePort	Puerto del servidor para la interfaz de usuario vía terminal.
ServerServiceMaximumConnections	Número máximo de conexiones simultáneas para la interfaz de usuario vía terminal.
ServerServiceTimeout	Caducidad de las conexiones para la interfaz de usuario vía terminal.
ServerHttpHostName	Nombre del servidor para la interfaz de usuario vía HTTP.
ServerHttpPort	Puerto del servidor para la interfaz de usuario vía HTTP.
ServerHttpMaximumConnections	Número máximo de conexiones simultáneas para la interfaz de usuario vía HTTP.
ServerHttpTimeout	Caducidad de las conexiones para la interfaz de usuario vía HTTP.
ServerEventHostName	Nombre del servidor para la entrada de mensajes de eventos.
ServerEventPort	Puerto del servidor para la entrada de mensajes de eventos.
ServerEventMaximumConnections	Número máximo de conexiones simultáneas para entrada de mensajes de eventos.
ServerEventTimeout	Caducidad de las conexiones para la entrada de mensajes de eventos.

Pruebas de corrección de fallas

Componentes utilizados para las pruebas generales

Para realizar las pruebas generales, utilizamos los siguientes componentes, en un total de 6 equipos de cómputo:

- **Equipo Gestor**: El sistema de corrección de fallas y correlación de eventos de manera automática. Un conjunto de reglas representativas que se encargarán de intentar solucionar los problemas reportados por el *script* de monitorización. Un conjunto de *scripts* que serán utilizados por las reglas para realizar diversas tareas de comunicación y operaciones en los equipos monitorizados.
- **Equipo Monitor:** *Script* que monitorizará y reportará la incidencia de problemas de tres equipos en cuanto a estado de la conexión de ese elemento en la red, memoria *RAM* disponible, espacio en disco duro y tráfico de red entrante.
- **Equipo Administrador de Tickets:** Un administrador de tickets al cual se le reportarán las fallas y el estado de su procesamiento.

- **Equipo administrado 1.**
- **Equipo administrado 2.**
- **Equipo administrado 3**.

Lo que queremos probar aquí es el correcto funcionamiento del sistema en general, así como sus capacidades en cuanto a la recepción de mensajes de eventos y su procesamiento de manera adecuada en un trabajo conjunto entre distintos sistemas, servidores, reglas, *scripts*, etc.

Monitor

Script:	monitor.sh
Descripción:	Es un monitor de elementos de red, que lee los atributos de cada elemento a partir de un archivo llamado *NES.TXT*. Verifica a cada segundo que haya conexión hacia el elemento a través de la ejecución del comando *ping*, que el servicio de apache esté levantado a través del comando *wget* y a través de conexiones vía *Secure Shell* solicita información del elemento de red, para conocer el espacio libre en *RAM*, disco duro y el flujo entrante de red. Si alguna de estas pruebas falla envía un mensaje de falla al gestor de fallas.
Nombre de Archivo:	NES.TXT
Descripción:	Archivo de configuración que describe los siguientes campos para cada elemento de la red: 1) la tarjeta de red del monitor por la que se comunicará al elemento de la red, 2) la *ip* del monitor en esa tarjeta, 3) el puerto por el cual enviará mensajes de falla al gestor de red, 4) la clave para la conexión *ssh*, 5) el usuario para la conexión *ssh*, 6) la *ip* a conectarse vía *ssh*, 7) la cantidad de memoria *RAM* en kilobytes mínima disponible en el elemento antes de enviar un mensaje de advertencia, 8) la ubicación del disco duro a observar, 9) la cantidad mínima de espacio libre disponible en kilobytes en el disco duro, 10) la tarjeta de red a la cual observará el tráfico de red entrante y 11) la cantidad máxima de tráfico entrante en bytes.

```
<TRedMonitors.sh> <IpNEM> <PuertoNEM> <PswNE> <UsrNE> <IpNE> <MinRAMks> <PathHDD> <MinHDDks> <Tar.RedNE> <MaxNetTrafficB>
eth0 127.0.0.1 8003 djtf dinamica 192.168.72.110 150000 /dev/sda1 98000000 eth0 1000
eth0 127.0.0.1 8003 sfed dinamica 192.168.72.111 150000 /dev/sda2 300000 eth1 1000
eth0 127.0.0.1 8003 jyrv dinamica 192.168.72.120 150000 /dev/sda5 79320000 eth0 1000
```

Codificación de procedimientos

Estableceremos 6 procedimientos representativos, que permiten en su conjunto, clasificar mensajes de eventos, corroborar fallas en la red y realizar tareas correctivas. A continuación se hará una descripción de cada procedimiento, en los apéndices se puede encontrar el código de los procedimientos y de los *scripts* utilizados en cada una de ellas.

Procedimiento: Unknown

Descripción:

Este procedimiento tiene como condición a la cadena vacía, lo que significa que cualquier mensaje de evento será manejado al llegar a ella pues no hay condición de entrada.
Su única tarea consiste en insertar el contenido del mensaje en un *ticket* en la base de datos para reportarlo como desconocido, es decir, que no existe algún procedimiento para darle tratamiento.

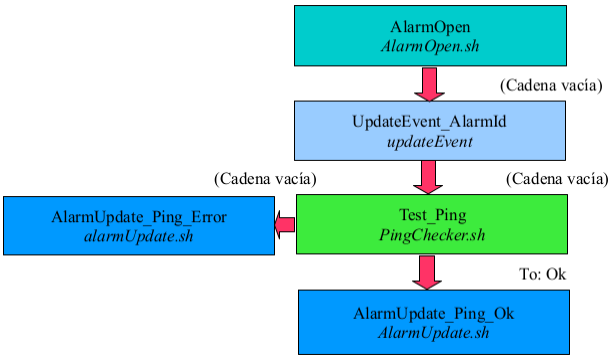
Diagrama:

Procedimiento: Unreachable

Descripción:

Inserta el contenido del mensaje de evento en un *ticket* de la base de datos y complementa el informe con el resultado de intentar comunicarse con el elemento descrito en el mensaje. No se cierra el *ticket* en ningún caso pues no se realiza ninguna acción correctiva.

Diagrama:

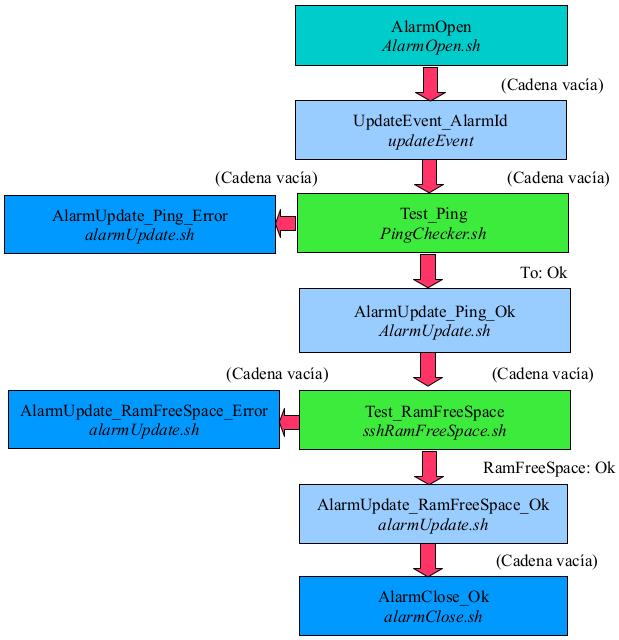


Procedimiento: RamFreeSpace

Descripción:

Inserta el contenido del mensaje de evento en un *ticket* en la base de datos, complementa el informe con el resultado de intentar comunicarse con el elemento descrito en el mensaje y obtener la cantidad de memoria *RAM* que se encuentra libre en ese momento, actualiza al *ticket* con esa información y si la cantidad de *RAM* libre es mayor a la mínima definida, entonces cierra al *ticket*.

Diagrama:

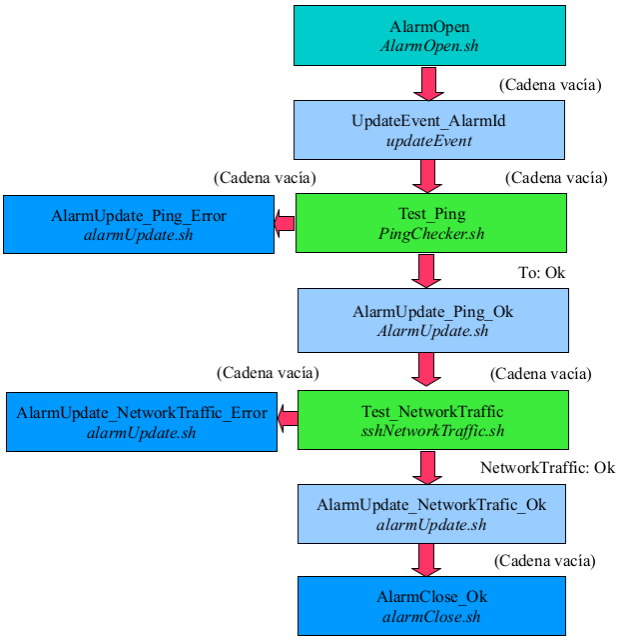


Procedimiento: NetworkTraffic

Descripción:

Inserta el contenido del mensaje de evento en un *ticket* en la base de datos, complementa el informe con el resultado de intentar comunicarse con el elemento descrito en el mensaje y obtener la cantidad de bytes que se están transfiriendo por segundo hacia el elemento descrito en el mensaje, actualiza al *ticket* con esa información y si el flujo de bytes por segundo es menor al máximo definido, entonces cierra al *ticket*.

Diagrama:

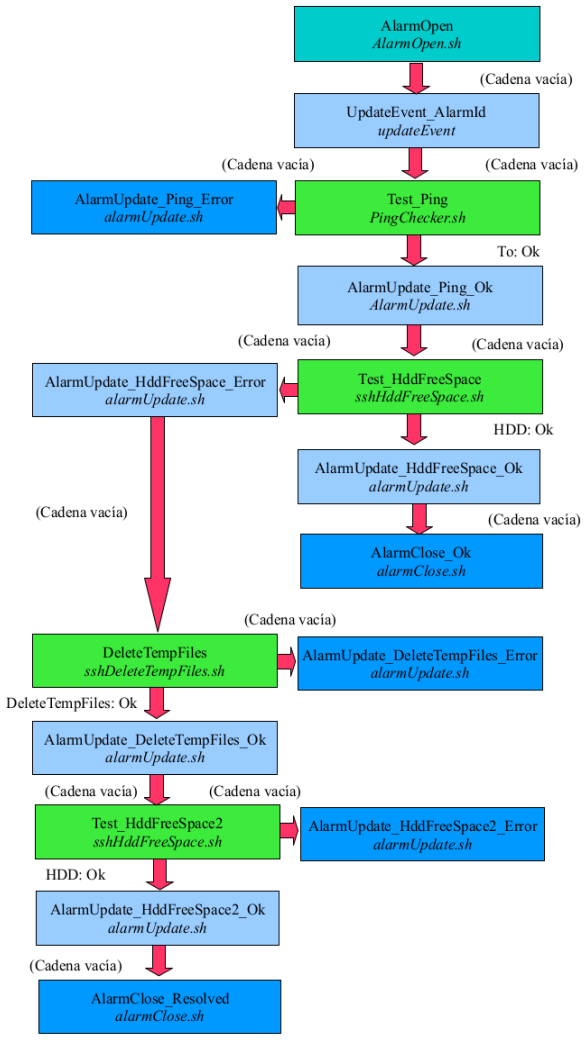


Procedimiento: HDDFreeSpace

Descripción:

Inserta el contenido del mensaje de evento en un *ticket* en la base de datos, complementa el informe con el resultado de intentar comunicarse con el elemento descrito en el mensaje y obtener la cantidad de kilobytes libres en el disco duro de el elemento descrito en el mensaje, actualiza al *ticket* con esa información y si el espacio libre es menor al mínimo definido, entonces borra los archivos temporales, si aún así el espacio libre es poco, entonces sólo actualiza al *ticket* con esa información, y si el espacio libre es suficiente actualiza al *ticket* y lo cierra.

Diagrama:

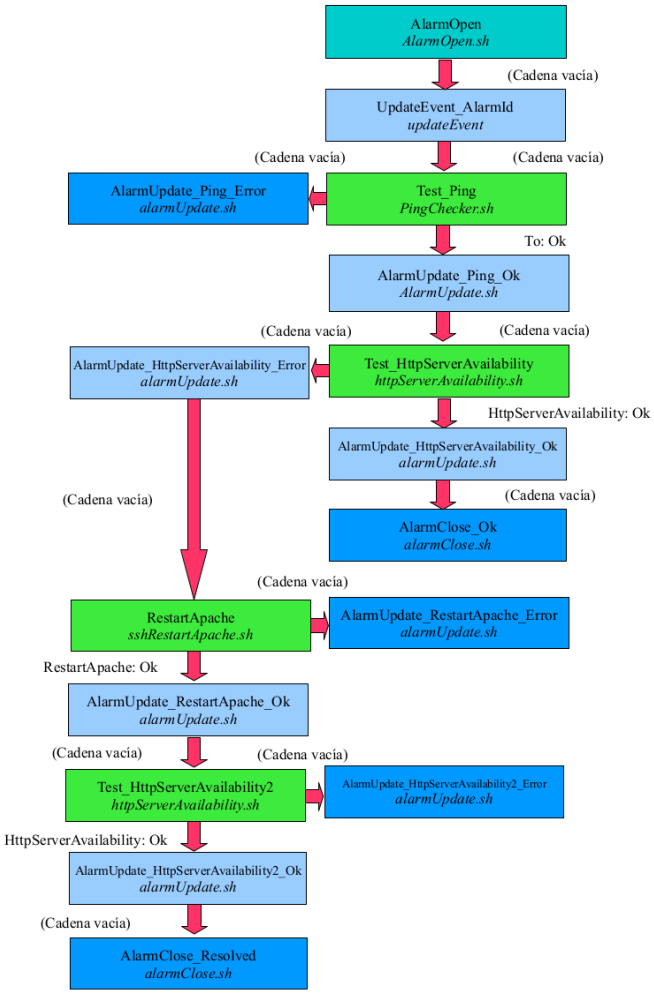


Procedimiento: HttpServerAvailability

Descripción:

Inserta el contenido del mensaje de evento en un *ticket* en la base de datos, complementa el informe con el resultado de intentar comunicarse con el elemento descrito en el mensaje y verificar que el servicio *Http* se encuentra disponible, actualiza al *ticket* con esa información y si el servicio no está activo, entonces intenta reiniciar el servicio, si aún así el servicio no está activo, entonces sólo actualiza al *ticket* con esa información, y si el servicio se encuentra disponible entonces actualiza al *ticket* y lo cierra.

Diagrama:



Guión

El guión que efectuamos fue el siguiente:

- Iniciamos al sistema gestor y le cargamos los procedimientos.
- Pruebas de funcionamiento:
  - Enviamos un mensaje del evento.
  - Verificamos que respete la clasificación del mensaje del evento de acuerdo a su contenido.
  - Verificamos que se el flujo del mensaje del evento sea el correcto.
  - Verificamos que la ejecución de los comandos y *scripts* sean adecuados.
- Utilizamos un programa generador de mensajes de eventos que nos permite establecer el número de mensajes que serán enviados al sistema gestor y evaluamos su desempeño a distintas cantidades de mensajes, en la recepción, almacenaje en el contenedor, desbordamiento a disco y procesamiento respetando el orden de llegada.

Para probar cada una de las reglas utilizamos distintos procedimientos:

- ***Unknown:*** Para probar este procedimiento enviamos un mensaje de evento que no fuera reconocible por ninguno de los procedimientos, para que así fuera reportado al administrador de *tickets*.
- ***Unreachable:*** Para probar este procedimiento enviamos un mensaje de evento falso de que se había perdido la comunicación con el elemento de la red, pero al ser procesado se comprobó el correcto funcionamiento del elemento y el *ticket* que se abrió se cerró anunciando esto. También se probó desconectando al equipo de la red, lo que dejaba al *ticket* abierto con la comprobación del problema por parte del gestor.
- ***RamFreeSpace:*** Se hicieron las pruebas que se efectuaron en el caso de *Unreachable*, y además se probó con mensajes de eventos falsos para la corroboración del buen funcionamiento y cierre del *ticket*, pero también saturamos a los equipos con procesos que utilizaran mucha memoria *RAM* para que se generaran mensajes de eventos reales y que al corroborar el problema se dejara al *ticket* abierto con el añadido de la corroboración del problema.
- ***NetworkTraffic:*** Se hicieron las pruebas que se efectuaron en el caso de *Unreachable*, y además se probó con mensajes de eventos falsos para la corroboración del buen funcionamiento y cierre del *ticket*, pero también saturamos a los equipos con procesos que incrementaran el tráfico entrante de red para que se generaran mensajes de eventos reales y que al corroborar el problema se dejara al *ticket* abierto con el añadido de la corroboración del problema.
- ***HDDFreeSpace:*** Se hicieron las pruebas que se efectuaron en el caso de *Unreachable*, y además se probó con mensajes de eventos falsos para la corroboración del buen funcionamiento y cierre del *ticket*, pero también saturamos a los equipos con archivos que incrementaran el uso de disco duro para que se generaran mensajes de eventos reales y que al corroborar el problema se intentara resolverlo, dejando el equipo listo para que en algunas veces se pudiera resolver y en otras no, todos estos procedimientos

se debían reflejar en el historial del *ticket*.

- ***HttpServerAvailability:*** Se hicieron las pruebas que se efectuaron en el caso de *Unreachable*, y además se probó con mensajes de eventos falsos para la corroboración del buen funcionamiento y cierre del *ticket*, pero también apagamos el servidor del equipo para que se generaran mensajes de eventos reales y que al corroborar el problema se intentara resolverlo, dejando el equipo listo para que en algunas veces se pudiera resolver y en otras no, todos estos procedimientos se debían reflejar en el historial del *ticket*.

Resultados

Realizamos este guión varias veces con cada uno de los procedimientos y se resolvieron satisfactoriamente, así como se podía revisar el historial del procesamiento de cada uno de los eventos en el sistema administrador de *tickets*. Así mismo, se realizaron pruebas de estrés, en donde se llegó a recibir de manera adecuada un aproximado de hasta 300 mensajes de eventos por segundo. El rendimiento en el procesamiento no es posible medirlo, pues depende de lo que tardan en ejecutarse los comandos que componen a cada procedimiento. Pero en el caso de insertar sólo mensajes de eventos que caen en el caso de *Unreachable*, todos los mensajes de eventos eran procesados al tiempo en que llegaban. También se verificó el correcto funcionamiento del contenedor y su desbordamiento en disco, así como se respetó el procesamiento de los mensajes de eventos de acuerdo al orden en el que llegaron.

Pruebas de correlación de eventos

Componentes utilizados para las pruebas de correlación

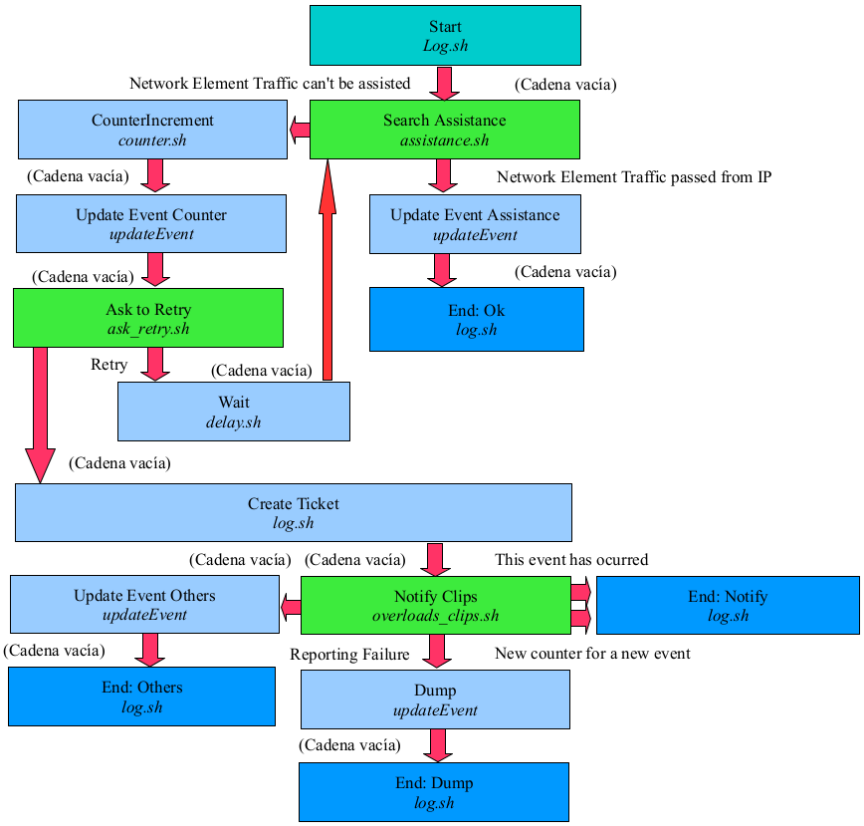
Para realizar las pruebas de correlación, utilizamos los siguientes componentes, en un sólo equipo de cómputo:

- **Equipo Gestor**: El sistema de corrección de errores y correlación de eventos de manera automática, cargado con un procedimiento que ofrece información de correlación que será alimentada a *CLIPS*. Un conjunto de *scripts* que serán utilizados por los procedimientos para realizar diversas tareas de comunicación y operaciones en los equipos monitorizados.
- **Equipo Generador de Mensajes de Eventos:** Programa que reportará la incidencia de problemas ficticios.
- **Equipo con *CLIPS***: En este equipo se encuentra la instancia de *CLIPS* encargada de realizar las correlaciones de eventos, al cual le será cargada una serie de sentencias de correlación de mensajes de eventos.

Lo que queremos probar aquí es el correcto funcionamiento en la correlación de eventos en la integración de Java con *CLIPS*.

Codificación de procedimientos

Estableceremos un procedimiento representativo, que permita entre otras cosas, el envío de información a *CLIPS*, para su correlación.

Procedimiento:	Overloads
Descripción: Este procedimiento tiene como condición a un evento de sobrecarga. Lo que hace es intentar desviar el tráfico a algún otro elemento de red, hará este intento 3 veces y en caso de no recibir ayuda notificará a la instancia de *CLIPS*. En caso de que *CLIPS* detecte que hay una gran cantidad de fallas con respecto a la misma *ip*, *CLIPS* responderá con un reporte de las ip's relacionadas a la misma *ip* hacia un *log* de eventos.	**Diagrama:** 

Archivo de sentencias cargado en CLIPS

Archivo:	Overloads.clp
Descripción:	Recibe información acerca de mensajes de eventos provenientes de alguna *ip*, en caso de haber reportados más de 2 eventos de falla de la misma *ip*, se despliega un reporte mostrando la *ip* y el número de veces que ha presentado el mensaje evento, así como el contador de ese evento se reiniciará a cero. En caso de ser la primera vez responderá avisando que ese mensaje de evento es la primera vez que ocurre. En caso de no ser la primera vez ni la tercera vez que ocurre, entonces sólo muestra el número de veces que se tiene conocimiento de ese evento de falla.

Guión

El guión que efectuamos fue el siguiente:

- Iniciamos al sistema gestor y se le carga el procedimiento.
- Iniciamos al servidor de *CLIPS* y se le carga el archivo de sentencias.
- Enviamos un mensaje de evento con una *ip* determinada.
- Verificamos su procesamiento y en caso de que no se haya podido desviar el tráfico, la *ip* sea cargada en la memoria de trabajo de *CLIPS*.
- Efectuamos los dos pasos anteriores probando con diferentes *ip's*, hasta que se repita alguna tres veces en la memoria de trabajo de *CLIPS* y verificamos si se hace el reporte por parte de *CLIPS* al *log* eventos.

Resultado

Realizamos este guión diez veces y el sistema lo realizó de manera adecuada.

Conclusiones de las pruebas

Aunque el rendimiento del sistema depende directamente de la complejidad de los *scripts* que componen a los procedimientos, también se hizo notoria la importancia de mejorar, en futuras líneas de investigación, la capacidad de recepción de los mensajes de eventos a través del servidor de recepción de mensajes de eventos. Más sin embargo, la recepción de hasta 300 mensajes de falla por segundo es una muy buena cantidad.

Capítulo IV

Futuras líneas de investigación y conclusiones

Sobre el futuro de este proyecto

Al determinar los alcances del proyecto y durante las pruebas, se identificaron varios puntos en los que se puede trabajar más adelante y se le dieron prioridad a centrales que permitieran generar un sistema mínimo funcional. A continuación mencionaremos los puntos en los que se puede realizar trabajos de investigación.

Integración dentro de un OSS o MOM

En el presente trabajo no abordamos las dificultades técnicas que supondría integrar al sistema inteligente dentro de un OSS o un MOM. Hay mucho trabajo por realizar dentro de esta línea, ya que dentro de una red de telecomunicaciones mixta en proveedores, los protocolos de comunicación son muy variados.

Seguridad

Este proyecto requiere de la implementación de diversos módulos que resguarden la seguridad de los datos, de manera que no sea vulnerable a ataques que puedan afectar su correcto funcionamiento.

Establecimiento de prioridades en el tratamiento de mensajes de eventos

El sistema actual no cuenta con un proceso que permita priorizar el tratamiento de los mensajes de eventos que requieran de una atención más urgente, para ello se puede pensar en un gestor ágil de datos que permita organizar los mensajes a partir de su nivel de prioridad.

Identificación de fallas raíz

Se antoja una línea de investigación para establecer una metodología para integrar de manera eficaz el conocimiento actual de los técnicos encargados de la administración de la red y por otra parte, establecer una metodología que permita el uso de diversas tecnologías dedicadas a la minería de datos, para utilizarlas sobre los históricos de los mensajes de eventos y que de esta manera se puedan descubrir patrones de comportamientos que son difíciles de identificar en cuanto a fallas raíz que

desencadenan a otras fallas, para así evitar que se vaya propagando el error por la red desde una etapa temprana. Y para esto existen dos perspectivas, una es la que corresponde a las fallas que existen y se identifican como fallas raíz, pero también existen las fallas que no son notificadas y que requieren del análisis de otras para realizar el diagnóstico de esta nueva falla que el sistema ha identificado y dado de alta de manera automática al no ser una falla explícita. Queda así claro que para el descubrimiento de fallas raíz se requiere del uso de dos perspectivas y posiblemente de dos métodos diferentes a modo de ser codificadas en el sistema para su identificación y tratamiento.

Capacidad de recepción de mensajes de eventos

Durante las pruebas, se identificó un cuello de botella en la recepción de mensajes de eventos, pues el sistema puede procesarlos más rápido que la velocidad en que los recibe. Se puede mejorar el rendimiento del sistema mejorando la velocidad en la recepción de mensajes.

Interfaz gráfica

Se pueden desarrollar diversas interfaces gráficas que facilitarían el uso del sistema, como puede ser: el desarrollo de las interfaces gráficas para la edición y administración de los procedimientos, monitores que permitan visualizar el estado del sistema, generadores de estadísticas, administradores de tareas e interfaces gráficas que permitan realizar simulaciones y pruebas.

Recuperación del sistema ante caídas

Otro elemento importante y nada trivial, es la enorme cantidad de programación requerida para los comportamientos no habituales del sistema, es decir la recuperación del sistema ante fallas de energía eléctrica o pérdida de datos, de modo que se pueda garantizar la integridad de los datos y el restablecimiento del sistema ante caídas.

Conclusiones sobre el sistema y el proyecto

Sobre los sistemas de gestión automática de fallas en las redes de telecomunicaciones

La automatización para la gestión de fallas en las redes de telecomunicaciones, es una tarea compleja, que integra distintas tecnologías y métodos. Pero vale la pena cuando una cantidad importante de fallas es atendida de manera automática, y así se puede aprovechar de mejor manera el tiempo de los técnicos en la investigación de nuevas y mejores técnicas de resolución de problemas.

Si bien es deseable lograr una automatización total, esto por ahora no es posible dada la gran cantidad de fallas diferentes que cada elemento de la red puede generar y peor aún, cuando las fallas en su combinación generan fallas nuevas de las cuales se complica a través de síntomas secundarios.

También es un reto, encontrar las configuraciones óptimas que nos permitan lograr el mejor rendimiento, pues depende del equipo donde se implemente el sistema, el tipo de problemas que se van a resolver y las tecnologías con las que se cuenten para las comunicaciones entre los elementos.

Sobre el proyecto

A continuación se enuncian los logros, ventajas y desventajas identificados durante el desarrollo de este proyecto:

- ***Logros:*** Se cumple el objetivo de contar con una plataforma que permita resolver problemas de los cuales ya se conoce el proceso de solución, y que esta plataforma pueda reutilizar la serie de *scripts* que utilizan los técnicos para el mantenimiento de la red de telecomunicaciones, administrándolos a través procedimientos que van realizando el diagnóstico del problema y formulando una serie soluciones hasta resolverlo o reportarlo a algún técnico con el historial de los procedimientos que fueron aplicados. Es también un gran logro, que se puedan recibir sin problemas, picos de más de 300 mensajes de eventos por segundo en un equipo de escritorio, y procesarse para resolver problemas reales, para luego almacenar en un sistema de *tickets* la serie de procedimientos que se aplicaron y en su caso, dejar el *ticket* de abierto en caso de que no se lograra resolver la falla de manera automática.
- ***Desventajas:*** La adquisición y codificación del conocimiento de los técnicos es una tarea muy difícil, por un lado el temor del técnico a perder valor ante la empresa, sin tener la visión de su utilidad en la resolución de problemas nuevos o la mejora de los procesos actuales, y por otro lado, mucho del trabajo que realizan los técnicos se ha adquirido de forma empírica, lo cual complica enormemente su transferencia. Lo que funciona es generar procedimientos de manera incremental a medida que se presentan los casos con el cuidado de no hacerlos simples y claros.
- ***Ventajas:*** Este sistema híbrido, combina lo mejor de dos mundos: el de los diagnósticos y acciones correctivas de las máquinas de estados, y el de los sistemas expertos para la correlación de eventos. Esto permite recibir una gran cantidad de mensajes de eventos que se van procesando a través de varias capas. Si no se contara con el contenedor de mensajes de eventos, el administrador de tareas no se daría abasto, si no se contara con el procesamiento de mensajes de eventos por máquinas de estados, el sistema experto se saturaría ante un flujo denso de nuevos hechos, y si no se contara con el sistema experto, sería muy complicada la correlación de eventos a través de las máquinas de estados. Cada módulo permite realizar de manera eficiente cada una de las tareas. Ante la competencia, como se menciona en este trabajo, algunos sistemas de gestión de fallas un cuentan con un sistema de correlación tan completo como el que cuenta *CLIPS* y otros no permiten la administración de fallas de elementos de red de otros proveedores, pero con este sistema se pueden administrar todos los equipos que cuenten con algún tipo de interfaz al que se pueda acceder desde el sistema operativo anfitrión en donde se instale el sistema de gestión de fallas.

Apéndice I

Códigos e implementaciones

Procedimientos utilizados en las pruebas de corrección de fallas

Procedimiento: Unknown

Condición: `(Cadena Vacía)`

Ejemplo de mensaje: `ERROR From: 192.168.72.105 To: 192.168.72.110`

Descripción: Este procedimiento tiene como condición a la cadena vacía, lo que significa que cualquier mensaje de evento será manejado al llegar a ella pues no hay condición de entrada.
Su única tarea consiste en insertar el contenido del mensaje en un *ticket* en la base de datos para reportarlo como desconocido, es decir, que no existe algún procedimiento para darle tratamiento.

Diagrama:

Código XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<finiteStateMachine name="Unknown" initialState="AlarmOpen" condition="">
          <states>
                    <state name="AlarmOpen" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmOpen.sh
                              ./pruebas/logs/UnknownEvents.log
                              &amp;quot;@&amp;quot;
                    " />
          </states>
</finiteStateMachine>
```

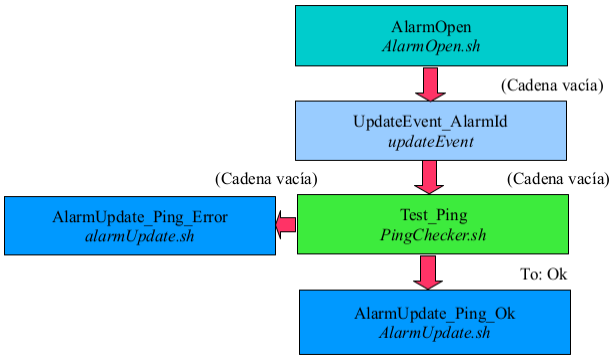
Procedimiento: Unreachable

Condición: `ERROR PING`

Ejemplo de mensaje: `ERROR PING From: 192.168.72.105 To: 192.168.72.110`

Descripción: Inserta el contenido del mensaje de evento en un *ticket* de la base de datos y complementa el informe con el resultado de intentar comunicarse con el elemento descrito en el mensaje. No se cierra el *ticket* en ningún caso pues no se realiza ninguna acción correctiva.

Diagrama:



Código XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<finiteStateMachine name="Unreachable" initialState="AlarmOpen" condition="ERROR PING">
          <states>
                    <state name="AlarmOpen" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmOpen.sh
                              ./pruebas/logs/Unreachable_AlarmOpen.log
                              &amp;quot;@&amp;quot;
                    " />
                    <state name="UpdateEvent_AlarmId" timeOut="10000" command="updateEvent" />
                    <state name="Test_Ping" timeOut="10000" command="bash
                              ./pruebas/scripts/pingChecker.sh
                              &amp;quot;@&amp;quot;
                    " />
                    <state name="AlarmUpdate_Ping_Ok" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmUpdate.sh
                              ./pruebas/logs/Unreachable_AlarmUpdate_Ping_Ok.log &amp;quot;@ Ping Ok.&amp;quot;
                    " />
                    <state name="AlarmUpdate_Ok" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmUpdate.sh
                              ./pruebas/logs/Unreachable_AlarmUpdate_Ok.log
                              &amp;quot;@&amp;quot;
                    " />
                    <state name="AlarmUpdate_Ping_Error" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmUpdate.sh
                              ./pruebas/logs/Unreachable_AlarmUpdate_Ping_Error.log
                              &amp;quot;@ Ping Error.&amp;quot;
                    " />
          </states>
          <transitions>
                    <transition from="AlarmOpen" to="UpdateEvent_AlarmId" condition="" />
                    <transition from="UpdateEvent_AlarmId" to="Test_Ping" condition="" />
                    <transition from="Test_Ping" to="AlarmUpdate_Ping_Ok" condition="To: Ok" />
                    <transition from="AlarmUpdate_Ping_Ok" to="AlarmUpdate_Ok" condition="" />
                    <transition from="Test_Ping" to="AlarmUpdate_Ping_Error" condition="" />
          </transitions>
</finiteStateMachine>
```

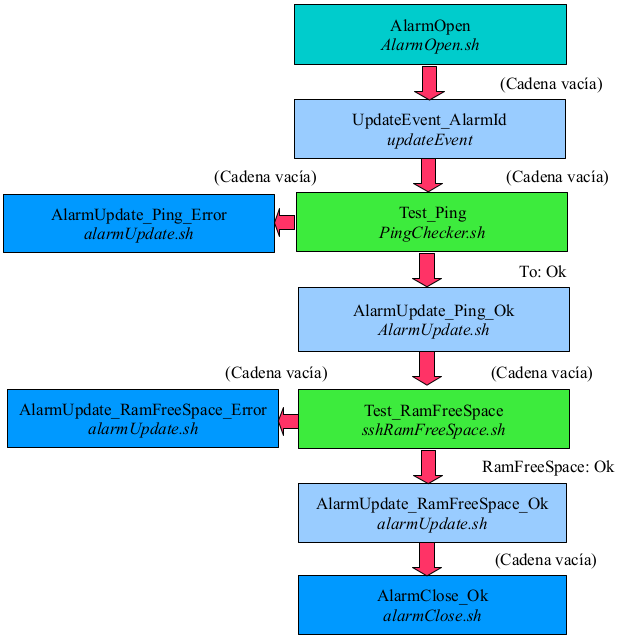
Procedimiento: RamFreeSpace

Condición: `WARNING RAM`

Ejemplo de mensaje: `WARNING RAM From: 192.168.72.105 To: 192.168.72.110`

Descripción: Inserta el contenido del mensaje de evento en un *ticket* en la base de datos, complementa el informe con el resultado de intentar comunicarse con el elemento descrito en el mensaje y obtener la cantidad de memoria *RAM* que se encuentra libre en ese momento, actualiza al *ticket* con esa información y si la cantidad de *RAM* libre es mayor a la mínima definida, entonces cierra al *ticket*.

Diagrama:



Código XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<finiteStateMachine name="RamFreeSpace" initialState="AlarmOpen" condition="WARNING RAM">
          <states>
                    <state name="AlarmOpen" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmOpen.sh
                              ./pruebas/logs/RamFreeSpace_AlarmOpen.log
                              &amp;quot;@&amp;quot;
                    " />
                    <state name="UpdateEvent_AlarmId" timeOut="10000" command="updateEvent" />
                    <state name="Test_Ping" timeOut="10000" command="bash
                              ./pruebas/scripts/pingChecker.sh
                              &amp;quot;@&amp;quot;
                    " />
                    <state name="AlarmUpdate_Ping_Ok" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmUpdate.sh
                              ./pruebas/logs/RamFreeSpace_AlarmUpdate_Ping_Ok.log
                              &amp;quot;@ Ping Ok.&amp;quot;
                    " />
                    <state name="Test_RamFreeSpace" timeOut="10000" command="bash
                              ./pruebas/scripts/sshRamFreeSpace.sh
                              &amp;quot;@&amp;quot;
                    "/>
                    <state name="AlarmUpdate_RamFreeSpace_Ok" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmUpdate.sh
                              ./pruebas/logs/RamFreeSpace_AlarmUpdate_RamFreeSpace_Ok.log
                              &amp;quot;@ RamFreeSpace Ok.&amp;quot;
                    " />
                    <state name="AlarmClose_Ok" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmClose.sh
                              ./pruebas/logs/RamFreeSpace_AlarmClose_Ok.log
                              &amp;quot;@&amp;quot;
                    " />
                    <state name="AlarmUpdate_Ping_Error" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmUpdate.sh
                              ./pruebas/logs/RamFreeSpace_AlarmUpdate_Ping_Error.log
                              &amp;quot;@ Ping Error.&amp;quot;
                    " />
                    <state name="AlarmUpdate_RamFreeSpace_Error" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmUpdate.sh
                              ./pruebas/logs/RamFreeSpace_AlarmUpdate_RamFreeSpace_Error.log
                              &amp;quot;@ RamFreeSpace Error.&amp;quot;
                    " />
          </states>
          <transitions>
                    <transition from="AlarmOpen" to="UpdateEvent_AlarmId" condition="" />
                    <transition from="UpdateEvent_AlarmId" to="Test_Ping" condition="" />
                    <transition from="Test_Ping" to="AlarmUpdate_Ping_Ok" condition="To: Ok" />
                    <transition from="AlarmUpdate_Ping_Ok" to="Test_RamFreeSpace" condition="" />
                    <transition from="Test_RamFreeSpace" to="AlarmUpdate_RamFreeSpace_Ok"
                              condition="RamFreeSpace: Ok" />
                    <transition from="AlarmUpdate_RamFreeSpace_Ok" to="AlarmClose_Ok" condition="" />
                    <transition from="Test_Ping" to="AlarmUpdate_Ping_Error" condition="" />
                    <transition from="Test_RamFreeSpace" to="AlarmUpdate_RamFreeSpace_Error"
                              condition="" />
```

```
        </transitions>
</finiteStateMachine>
```

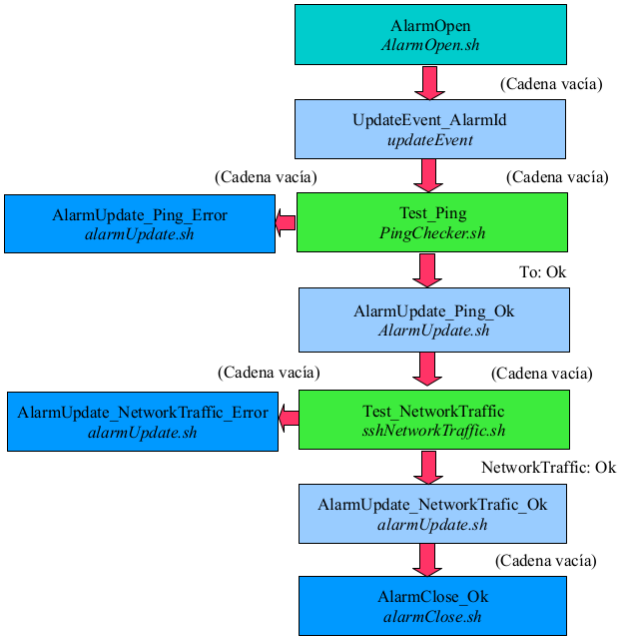
Procedimiento: NetworkTraffic

Condición: `WARNING NETWORK TRAFFIC`

Ejemplo de mensaje: `WARNING NETWORK TRAFFIC From: 192.168.72.105 To: 192.168.72.110`

Descripción: Inserta el contenido del mensaje de evento en un *ticket* en la base de datos, complementa el informe con el resultado de intentar comunicarse con el elemento descrito en el mensaje y obtener la cantidad de bytes que se están transfiriendo por segundo hacia el elemento descrito en el mensaje, actualiza al *ticket* con esa información y si el flujo de bytes por segundo es menor al máximo definido, entonces cierra al *ticket*.

Diagrama:



Código XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<finiteStateMachine name="NetworkTraffic" initialState="AlarmOpen"
         condition="WARNING NETWORK TRAFFIC">
         <states>
                  <state name="AlarmOpen" timeOut="10000" command="bash
                           ./pruebas/scripts/alarmOpen.sh
                           ./pruebas/logs/NetworkTraffic_AlarmOpen.log
                           &amp;quot;@&amp;quot;
                  " />
                  <state name="UpdateEvent_AlarmId" timeOut="10000" command="updateEvent" />
                  <state name="Test_Ping" timeOut="10000" command="bash
                           ./pruebas/scripts/pingChecker.sh
                           &amp;quot;@&amp;quot;
                  " />
                  <state name="AlarmUpdate_Ping_Ok" timeOut="10000" command="bash
                           ./pruebas/scripts/alarmUpdate.sh
                           ./pruebas/logs/NetworkTraffic_AlarmUpdate_Ping_Ok.log
                           &amp;quot;@ Ping Ok.&amp;quot;
                  " />
                  <state name="Test_NetworkTraffic" timeOut="10000" command="bash
                           ./pruebas/scripts/sshNetworkTraffic.sh
                           &amp;quot;@&amp;quot;
                  " />
                  <state name="AlarmUpdate_NetworkTraffic_Ok" timeOut="10000" command="bash
                           ./pruebas/scripts/alarmUpdate.sh
                           ./pruebas/logs/NetworkTraffic_AlarmUpdate_NetworkTraffic_Ok.log
                           &amp;quot;@ NetworkTraffic Ok.&amp;quot;
```

```
                " />
                <state name="AlarmClose_Ok" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmClose.sh
                          ./pruebas/logs/NetworkTraffic_AlarmClose_Ok.log
                          &quot;@&quot;
                " />
                <state name="AlarmUpdate_Ping_Error" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmUpdate.sh
                          ./pruebas/logs/NetworkTraffic_AlarmUpdate_Ping_Error.log
                          &quot;@ Ping Error.&quot;
                " />
                <state name="AlarmUpdate_NetworkTraffic_Error" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmUpdate.sh
                          ./pruebas/logs/NetworkTraffic_AlarmUpdate_NetworkTraffic_Error.log
                          &quot;@ NetworkTraffic Error.&quot;
                " />
        </states>
        <transitions>
                <transition from="AlarmOpen" to="UpdateEvent_AlarmId" condition="" />
                <transition from="UpdateEvent_AlarmId" to="Test_Ping" condition="" />
                <transition from="Test_Ping" to="AlarmUpdate_Ping_Ok" condition="To: Ok" />
                <transition from="AlarmUpdate_Ping_Ok" to="Test_NetworkTraffic" condition="" />
                <transition from="Test_NetworkTraffic" to="AlarmUpdate_NetworkTraffic_Ok"
                          condition="NetworkTraffic: Ok" />
                <transition from="AlarmUpdate_NetworkTraffic_Ok" to="AlarmClose_Ok" condition="" />
                <transition from="Test_Ping" to="AlarmUpdate_Ping_Error" condition="" />
                <transition from="Test_NetworkTraffic" to="AlarmUpdate_NetworkTraffic_Error"
                          condition="" />
        </transitions>
</finiteStateMachine>
```

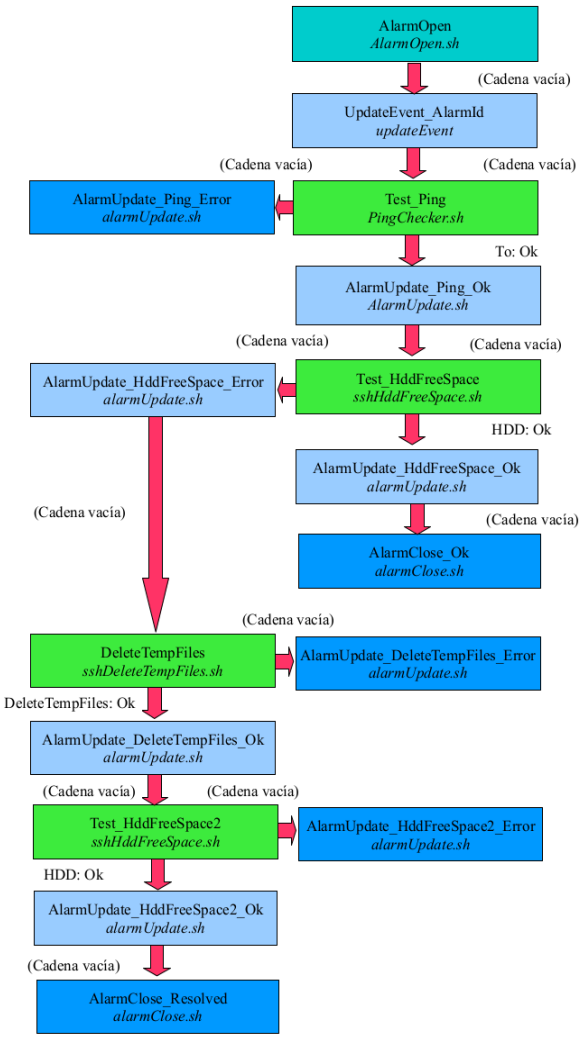
Procedimiento: HDDFreeSpace

Condición: `WARNING HDD`

Ejemplo de mensaje: `WARNING HDD From: 192.168.72.105 To: 192.168.72.110`

Descripción: Inserta el contenido del mensaje de evento en un *ticket* en la base de datos, complementa el informe con el resultado de intentar comunicarse con el elemento descrito en el mensaje y obtener la cantidad de kilobytes libres en el disco duro de el elemento descrito en el mensaje, actualiza al *ticket* con esa información y si el espacio libre es menor al mínimo definido, entonces borra los archivos temporales, si aún así el espacio libre es poco, entonces sólo actualiza al *ticket* con esa información, y si el espacio libre es suficiente actualiza al *ticket* y lo cierra.

Diagrama:



Código XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<finiteStateMachine name="HDDFreeSpace" initialState="AlarmOpen" condition="WARNING HDD">
          <states>
                    <state name="AlarmOpen" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmOpen.sh
                              ./pruebas/logs/HDDFreeSpace_AlarmOpen.log
                              &amp;quot;@&amp;quot;
                    " />
                    <state name="UpdateEvent_AlarmId" timeOut="10000" command="updateEvent" />
                    <state name="Test_Ping" timeOut="10000" command="bash
                              ./pruebas/scripts/pingChecker.sh
                              &amp;quot;@&amp;quot;
                    " />
                    <state name="AlarmUpdate_Ping_Ok" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmUpdate.sh
                              ./pruebas/logs/HDDFreeSpace_AlarmUpdate_Ping_Ok.log
                              &amp;quot;@ Ping Ok.&amp;quot;
                    " />
                    <state name="Test_HddFreeSpace" timeOut="10000" command="bash
                              ./pruebas/scripts/sshHddFreeSpace.sh
                              &amp;quot;@&amp;quot;
                    " />
                    <state name="AlarmUpdate_HddFreeSpace_Ok" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmUpdate.sh
                              ./pruebas/logs/HDDFreeSpace_AlarmUpdate_HddFreeSpace_Ok.log
                              &amp;quot;@ HDD Space Ok.&amp;quot;
                    " />
                    <state name="AlarmClose_Ok" timeOut="10000" command="bash
                              ./pruebas/scripts/alarmClose.sh
                              ./pruebas/logs/HDDFreeSpace_AlarmClose_Ok.log
                              &amp;quot;@&amp;quot;
```

```
                " />
                <state name="AlarmUpdate_Ping_Error" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmUpdate.sh
                          ./pruebas/logs/HDDFreeSpace_AlarmUpdate_Ping_Error.log
                          &quot;@ Ping Error.&quot;
                " />
                <state name="AlarmUpdate_HddFreeSpace_Error" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmUpdate.sh
                          ./pruebas/logs/HDDFreeSpace_AlarmUpdate_HddFreeSpace_Error.log
                          &quot;@ HDD Free Space Error.&quot;
                " />
                <state name="DeleteTempFiles" timeOut="10000" command="bash
                          ./pruebas/scripts/sshDeleteTempFiles.sh
                          &quot;@&quot;
                " />
                <state name="AlarmUpdate_DeleteTempFiles_Ok" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmUpdate.sh
                          ./pruebas/logs/HDDFreeSpace_AlarmUpdate_DeleteTempFiles_Ok.log
                          &quot;@ Delete Temp Files Ok.&quot;
                " />
                <state name="Test_HddFreeSpace2" timeOut="10000" command="bash
                          ./pruebas/scripts/sshHddFreeSpace.sh
                          &quot;@&quot;
                " />
                <state name="AlarmUpdate_HddFreeSpace2_Ok" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmUpdate.sh
                          ./pruebas/logs/HDDFreeSpace_AlarmUpdate_HddFreeSpace2_Ok.log
                          &quot;@ HDD Space Ok.&quot;
                " />
                <state name="AlarmClose_Resolved" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmClose.sh
                          ./pruebas/logs/HDDFreeSpace_AlarmClose_Resolved_Ok.log
                          &quot;@&quot;
                " />
                <state name="AlarmUpdate_DeleteTempFiles_Error" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmUpdate.sh
                          ./pruebas/logs/HDDFreeSpace_AlarmUpdate_DeleteTempFiles_Error.log
                          &quot;@ Delete Temporary Files Error.&quot;
                " />
                <state name="AlarmUpdate_HddFreeSpace2_Error" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmUpdate.sh
                          ./pruebas/logs/HDDFreeSpace_AlarmUpdate_HddFreeSpace2_Error.log
                          &quot;@ HDD Free Space Error 2.&quot;
                " />
        </states>
        <transitions>
                <transition from="AlarmOpen" to="UpdateEvent_AlarmId" condition="" />
                <transition from="UpdateEvent_AlarmId" to="Test_Ping" condition="" />
                <transition from="Test_Ping" to="AlarmUpdate_Ping_Ok" condition="To: Ok" />
                <transition from="AlarmUpdate_Ping_Ok" to="Test_HddFreeSpace"
                          condition="" />
                <transition from="Test_HddFreeSpace" to="AlarmUpdate_HddFreeSpace_Ok"
                          condition="HDD: Ok" />
                <transition from="AlarmUpdate_HddFreeSpace_Ok" to="AlarmClose_Ok"
                          condition="" />
                <transition from="Test_Ping" to="AlarmUpdate_Ping_Error" condition="" />
                <transition from="Test_HddFreeSpace" to="AlarmUpdate_HddFreeSpace_Error"
                          condition="" />
                <transition from="AlarmUpdate_HddFreeSpace_Error" to="DeleteTempFiles"
                          condition="" />
                <transition from="DeleteTempFiles" to="AlarmUpdate_DeleteTempFiles_Ok"
                          condition="DeleteTempFiles: Ok" />
                <transition from="AlarmUpdate_DeleteTempFiles_Ok" to="Test_HddFreeSpace2"
                          condition="" />
                <transition from="Test_HddFreeSpace2" to="AlarmUpdate_HddFreeSpace2_Ok"
                          condition="HDD: Ok" />
                <transition from="AlarmUpdate_HddFreeSpace2_Ok" to="AlarmClose_Resolved"
                          condition="" />
                <transition from="DeleteTempFiles" to="AlarmUpdate_DeleteTempFiles_Error"
                          condition="" />
                <transition from="Test_HddFreeSpace2" to="AlarmUpdate_HddFreeSpace2_Error"
                          condition="" />
        </transitions>
</finiteStateMachine>
```

Procedimiento: HttpServerAvailability

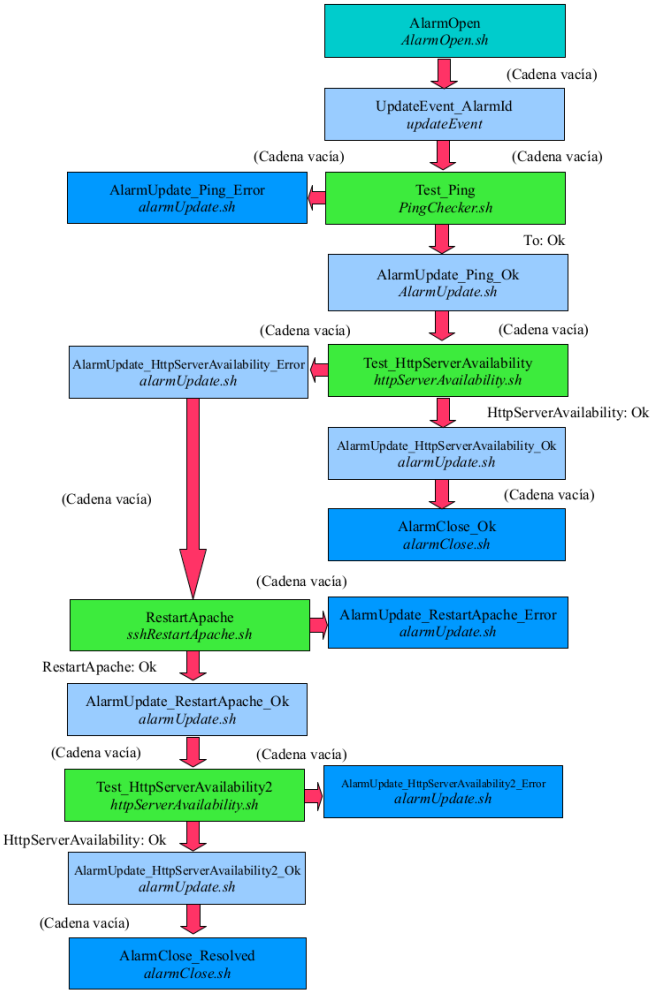
Condición: `ERROR HTTP`

Ejemplo de mensaje: `ERROR HTTP From: 192.168.72.105 To: 192.168.72.110`

Descripción: Inserta el contenido del mensaje de evento en un *ticket* en la base de datos, complementa el informe con el resultado de intentar comunicarse con el elemento descrito en el mensaje y verificar que el servicio *Http* se encuentra disponible, actualiza al *ticket* con esa información y si el servicio no está activo, entonces intenta reiniciar el servicio, si aún así el servicio no está activo, entonces sólo actualiza al *ticket* con esa información, y si el servicio se encuentra

disponible entonces actualiza al *ticket* y lo cierra.

Diagrama:



Código XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<finiteStateMachine name="HttpServerAvailability" initialState="AlarmOpen"
         condition="ERROR HTTP">
         <states>
                  <state name="AlarmOpen" timeOut="10000" command="bash
                           ./pruebas/scripts/alarmOpen.sh
                           ./pruebas/logs/HttpServerAvailability_AlarmOpen.log
                           &amp;quot;@&amp;quot;
                  " />
                  <state name="UpdateEvent_AlarmId" timeOut="10000" command="updateEvent" />
                  <state name="Test_Ping" timeOut="10000" command="bash
                           ./pruebas/scripts/pingChecker.sh
                           &amp;quot;@&amp;quot;
                  " />
                  <state name="AlarmUpdate_Ping_Ok" timeOut="10000" command="bash
                           ./pruebas/scripts/alarmUpdate.sh
                           ./pruebas/logs/HttpServerAvailability_AlarmUpdate_Ping_Ok.log
                           &amp;quot;@ Ping Ok.&amp;quot;
                  " />
                  <state name="Test_HttpServerAvailability" timeOut="10000" command="bash
                           ./pruebas/scripts/httpServerAvailability.sh
                           &amp;quot;@&amp;quot;
                  " />
                  <state name="AlarmUpdate_HttpServerAvailability_Ok" timeOut="10000" command="bash
                           ./pruebas/scripts/alarmUpdate.sh
                           ./pruebas/logs/
                                    HttpServerAvailability_AlarmUpdate_HttpServerAvailability_Ok.log
                           &amp;quot;@ HttpServerAvailability Ok.&amp;quot;
                  " />
```

```
                <state name="AlarmClose_Ok" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmClose.sh
                          ./pruebas/logs/HttpServerAvailability_AlarmClose_Ok.log
                          &amp;quot;@&amp;quot;
                " />
                <state name="AlarmUpdate_Ping_Error" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmUpdate.sh
                          ./pruebas/logs/HttpServerAvailability_AlarmUpdate_Ping_Error.log
                          &amp;quot;@ Ping Error.&amp;quot;
                " />
                <state name="AlarmUpdate_HttpServerAvailability_Error" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmUpdate.sh
                          ./pruebas/logs/
                                    HttpServerAvailability_AlarmUpdate_HttpServerAvailability_Error.log
                          &amp;quot;@ HttpServerAvailability Error.&amp;quot;
                " />
                <state name="RestartApache" timeOut="10000" command="bash
                          ./pruebas/scripts/sshRestartApache.sh
                          &amp;quot;@&amp;quot;
                " />
                <state name="AlarmUpdate_RestartApache_Ok" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmUpdate.sh
                          ./pruebas/logs/HttpServerAvailability_AlarmUpdate_RestartApache_Ok.log
                          &amp;quot;@ Restart Apache Ok.&amp;quot;
                " />
                <state name="Test_HttpServerAvailability2" timeOut="10000" command="bash
                          ./pruebas/scripts/httpServerAvailability.sh
                          &amp;quot;@&amp;quot;
                " />
                <state name="AlarmUpdate_HttpServerAvailability2_Ok" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmUpdate.sh
                          ./pruebas/logs/
                                    HttpServerAvailability_AlarmUpdate_HttpServerAvailability2_Ok.log
                          &amp;quot;@ HttpServerAvailability Ok.&amp;quot;
                " />
                <state name="AlarmClose_Resolved" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmClose.sh
                          ./pruebas/logs/HttpServerAvailability_AlarmClose_Resolved_Ok.log
                          &amp;quot;@&amp;quot;
                " />
                <state name="AlarmUpdate_RestartApache_Error" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmUpdate.sh
                          ./pruebas/logs/HttpServerAvailability_AlarmUpdate_RestartApache_Error.log
                          &amp;quot;@ RestartApache Error.&amp;quot;
                " />
                <state name="AlarmUpdate_HttpServerAvailability2_Error" timeOut="10000" command="bash
                          ./pruebas/scripts/alarmUpdate.sh
                          ./pruebas/logs/
                                    HttpServerAvailability_AlarmUpdate_HttpServerAvailability2_Error.log
                          &amp;quot;@ HttpServerAvailability Error 2.&amp;quot;
                " />
        </states>
        <transitions>
                <transition from="AlarmOpen" to="UpdateEvent_AlarmId" condition="" />
                <transition from="UpdateEvent_AlarmId" to="Test_Ping" condition="" />
                <transition from="Test_Ping" to="AlarmUpdate_Ping_Ok"
                          condition="To: Ok" />
                <transition from="AlarmUpdate_Ping_Ok" to="Test_HttpServerAvailability"
                          condition="" />
                <transition from="Test_HttpServerAvailability"
                          to="AlarmUpdate_HttpServerAvailability_Ok"
                          condition="HttpServerAvailability: Ok" />
                <transition from="AlarmUpdate_HttpServerAvailability_Ok" to="AlarmClose_Ok"
                          condition="" />
                <transition from="Test_Ping" to="AlarmUpdate_Ping_Error" condition="" />
                <transition from="Test_HttpServerAvailability"
                          to="AlarmUpdate_HttpServerAvailability_Error"
                          condition="" />
                <transition from="AlarmUpdate_HttpServerAvailability_Error"
                          to="RestartApache"
                          condition="" />
                <transition from="RestartApache" to="AlarmUpdate_RestartApache_Ok"
                          condition="RestartApache: Ok" />
                <transition from="AlarmUpdate_RestartApache_Ok"
                          to="Test_HttpServerAvailability2"
                          condition="" />
                <transition from="Test_HttpServerAvailability2"
                          to="AlarmUpdate_HttpServerAvailability2_Ok"
                          condition="HttpServerAvailability: Ok" />
                <transition from="AlarmUpdate_HttpServerAvailability2_Ok"
                          to="AlarmClose_Resolved"
                          condition="" />
                <transition from="RestartApache" to="AlarmUpdate_RestartApache_Error"
                          condition="" />
                <transition from="Test_HttpServerAvailability2"
                          to="AlarmUpdate_HttpServerAvailability2_Error"
                          condition="" />
        </transitions>
</finiteStateMachine>
```

Scripts utilizados en las pruebas de corrección de fallas

Script: updateEvent

Ejemplo de parámetros: `“ New Data To Append”`

Salida en caso de éxito: `ERROR From: 192.168.72.105 To: 192.168.72.110 New Data To Append`

Salida en caso de error: `ERROR From: 192.168.72.105 To: 192.168.72.110`

Descripción: Agrega información al mensaje de falla que transita por la máquina de estados.

Código Bash Script: No existe código del *script*, porque es una palabra reservada dentro del sistema.

Script: AlarmClose.sh

Ejemplo de parámetros: `“ERROR HTTP From: 192.168.72.105 To: 192.168.72.110 AlarmId: 23”`

Salida en caso de éxito: `AlarmClose: Ok AlarmId: 23`

Salida en caso de error: `AlarmClose: Error AlarmId: 23`

Descripción: Cierra un *ticket* al tomar el *AlarmId* de la alarma de un mensaje.

Código Bash Script:

```
#!/bin/bash
URL="http://ifm-ats.local"
DATA=$*
DATA=${DATA//\"/}
NAME=$1
LENGTH=${#NAME}
MESSAGE=${DATA:LENGTH+1}
ALARM_FIELD=`echo $MESSAGE | grep -o "AlarmId: [0-9]*"`
ALARM_ID=`echo $ALARM_FIELD | grep -o "[0-9]*"`
OUTPUT=`python ./pruebas/scripts/ats.py "$URL" --clear-alarm $ALARM_ID`
if(test $? -ne 0) then
          echo `date` "AlarmClose: Error AlarmId: $ALARM_ID" >> $1
          echo "AlarmClose: Error AlarmId: $ALARM_ID"
else
          echo "AlarmClose: Ok AlarmId: $ALARM_ID"
fi
```

Script: AlarmOpen.sh

Ejemplo de parámetros: `“ERROR HTTP From: 192.168.72.105 To: 192.168.72.110”`

Salida en caso de éxito: `ERROR HTTP From: 192.168.72.105 To: 192.168.72.110 AlarmId: -1`

Salida en caso de error: `ERROR HTTP From: 192.168.72.105 To: 192.168.72.110 AlarmId: 23`

Descripción: Abre un *ticket* y le asigna el mensaje de falla, devolviendo el *AlarmId* del *ticket*.

Código Bash Script:

```
#!/bin/bash
URL="http://ifm-ats.local"
DATA=$*
DATA=${DATA//\"/}
NAME=$1
LENGTH=${#NAME}
MESSAGE=${DATA:LENGTH+1}
OUTPUT=`python ./pruebas/scripts/ats.py $URL --create-alarm "$MESSAGE"`
if(test $? -ne 0) then
          echo `date` "$MESSAGE" >> $1
          echo "$MESSAGE AlarmId: -1"
else
          echo "$MESSAGE AlarmId: $OUTPUT"
fi
```

Script: AlarmUpdate.sh

Ejemplo de parámetros: `“ERROR HTTP From: 192.168.72.105 To: 192.168.72.110 AlarmId: 23”`

Salida en caso de éxito: `AlarmUpdate: Ok AlarmId: 23`

Salida en caso de error: `AlarmUpdate: Error AlarmId: 23`

Descripción: Añade información a un *ticket* a partir de su *AlarmId*.

Código Bash Script:

```
#!/bin/bash
URL="http://ifm-ats.local"
DATA=$*
DATA=${DATA//\"/}
NAME=$1
LENGTH=${#NAME}
MESSAGE=${DATA:LENGTH+1}
ALARM_FIELD=`echo $MESSAGE | grep -o "AlarmId: [0-9]*"`
ALARM_ID=`echo $ALARM_FIELD | grep -o "[0-9]*"`
OUTPUT=`python ./pruebas/scripts/ats.py "$URL" --comment-alarm $ALARM_ID "$MESSAGE"`
if(test $? -ne 0) then
          echo `date` "AlarmUpdate: Error AlarmId: $ALARM_ID" >> $1
          echo "AlarmUpdate: Error AlarmId: $ALARM_ID"
else
          echo "AlarmUpdate: Ok AlarmId: $ALARM_ID"
fi
```

Script: Ats.py

Ejemplo de parámetros:

```
Cerrar alarma: "http://ifm-ats.local" --clear-alarm 23

Crear alarma: "http://ifm-ats.local" --create-alarm "ERROR HTTP From:
192.168.72.105 To: 192.168.72.110"

Actualizar alarma: "http://ifm-ats.local" --comment-alarm 23 "ERROR HTTP From:
192.168.72.105 To: 192.168.72.110"
```

Salida en caso de éxito: `(Estatus de salida en sistema 0 o el AlarmID)`

Salida en caso de error: `(Estatus de salida en sistema 1)`

Descripción: Cierra, crea y actualiza *tickets* dentro de un gestor de *tickets* en una base de datos.

Código Bash Script:

```
#!/usr/bin/python

import sys
import httplib2
import simplejson as json
from pprint import *

http = httplib2.Http()

def do_http_request(service_url,
          request_type='GET',
          message='',
          token='',
          cookie=''):
    headersList = {'Content-type': 'application/json'}
    if token:
        headersList["X-CSRF-Token"] = token
    if cookie:
        headersList["Cookie"] = cookie
    #print json.dumps(headersList)
    return http.request(
        service_url,
        request_type,
        message,
        headers=headersList
    )

def do_login(username, password, token, url):
    login_service_url = url + '/restapi/user/login.json'
    login_message = '''
    {
        "username": "''' + username + '''",
        "password": "''' + password + '''"
```

```
    }
    '''
    response, content = do_http_request(login_service_url,
          'POST',
          login_message,
          token)
    return response["set-cookie"]

def get_services_session_token(cookie, url):
    session_service_url = url + '/services/session/token'
    response, content = do_http_request(session_service_url,
          'GET',
          '',
          '',
          cookie)
    return content

def get_nodes(token, cookie, url):
    node_service_url = url + '/restapi/node.json'
    return do_http_request(node_service_url, 'GET', '', token, cookie)

def close_ticket(ticket_id, token, cookie, url):
    node_service_url = url + '/restapi/node/' + ticket_id
    node_message = '''
    {
        "field_estado": { "und": { "value": "closed" } }
    }
    '''
    return do_http_request(node_service_url, 'PUT', node_message, token, cookie)

def clear_alarm(alarm_id, token, cookie, url):
    node_service_url = url + '/restapi/node/' + alarm_id
    node_message = '''
    {
        "field_alarma_estado": { "und": { "value": "inactive" } }
    }
    '''
    return do_http_request(node_service_url, 'PUT', node_message, token, cookie)

def add_page(token, cookie, url):
    node_service_url = url + '/restapi/node.json'
    node_message = '''
    {
        "title": "Test title",
        "body": { "und": [ { "value": "Test Body" } ] },
        "type": "page"
    }
    '''
    return do_http_request(node_service_url,
          'POST',
          node_message,
          token,
          cookie)

def add_alarm(alaramText, token, cookie, url):
    node_service_url = url + '/restapi/node.json'
    node_message = '''
    {
        "body": { "und": [ { "value": "''' + alaramText + '''" } ] },
        "type": "alarma"
    }
    '''
    response, content = do_http_request(node_service_url,
          'POST',
          node_message,
          token,
          cookie)
    if response['status'] == '200':
        return content
    return '{}'


def add_ticket(ticketText, alarmID, token, cookie, url):
    node_service_url = url + '/restapi/node.json'
    if alarmID == '':
        node_message = '''
        {
            "body": { "und": [ { "value": "''' + ticketText + '''" } ] },
            "type": "ticket"
        }
        '''
    else:
        #"field_alarm": { "und": [ { "target_id": "''' + alarmID + '''" } ] }
        node_message = '''
        {
            "body": { "und": [ { "value": "''' + ticketText + '''" } ] },
            "type": "ticket",
            "field_alarm": { "und": [ { "target_id": "alarm (''' + alarmID +
        ''')" } ] }
        }
        '''
    response, content = do_http_request(node_service_url,
          'POST',
          node_message,
          token,
```

```
          cookie)
    if response['status'] == '200':
        return content

    return '{}'

def comment_node(nodeID, comment_text, token, cookie, url):
    comment_service_url = url + '/restapi/comment.json'
    comment_message = '''
    {
      "nid":"''' + nodeID + '''",
      "subject":"Update",
      "comment_body":{
        "und":[{
          "value":"''' + comment_text + '''"
        }]
      }
    }
    '''

    response, content = do_http_request(comment_service_url,
          'POST',
          comment_message,
          token,
          cookie)
    if response['status'] == '200':
        return content

    return '{}'

def print_usage():
    print 'usage: ServiceURL --create-alarm "Alarm text"'
    print 'usage: ServiceURL --clear-alarm AlarmID'
    print 'usage: ServiceURL --create-ticket "Ticket text" [AlarmID]'
    print 'usage: ServiceURL --close-ticket TicketID'
    print 'usage: ServiceURL --comment-alarm AlarmID "Comment text"'
    print 'usage: ServiceURL --comment-ticket TicketID "Comment text"'


def main():
    args = sys.argv[1:]
    if not args:
        print_usage()
        sys.exit(1)

    # Take first argument as service base URL
    url = args[0]
    del args[0]

    param_create_alarm = False
    param_clear_alarm = False
    param_create_ticket = False
    param_close_ticket = False
    param_comment_alarm = False
    param_comment_ticket = False
    if args[0] == '--create-alarm':
        param_create_alarm = True
    elif args[0] == '--clear-alarm':
        param_clear_alarm = True
    elif args[0] == '--create-ticket':
        param_create_ticket = True
    elif args[0] == '--close-ticket':
        param_close_ticket = True
    elif args[0] == '--comment-alarm':
        param_comment_alarm = True
    elif args[0] == '--comment-ticket':
        param_comment_ticket = True
    else:
        print 'Error: Wrong command parameter.'
        print_usage()
        sys.exit(1)
    del args[0]

    token = get_services_session_token('', url)
    cookie = do_login('timsdevlab', '5ysUj3Df38qtEGzG', token, url)
    newToken = get_services_session_token(cookie, url)

    if param_create_alarm:
        alarm_text = args[0]
        #print 'Creating a new Alarm with:'
        #print 'Alarm text:', alarm_text
        alarm_json = add_alarm(alarm_text, newToken, cookie, url)
        alarm = json.loads(alarm_json)
        print alarm['nid']
    elif param_clear_alarm:
        alarm_id = args[0]
        #print 'Clearing an Alarm with:'
        #print 'Alarm ID:', alarm_id
        clear_alarm(alarm_id, newToken, cookie, url)
    elif param_create_ticket:
        ticket_text = args[0]
        del args[0]
        if args:
            alarm_id = args[0]
        else:
```

```
            alarm_id = ''
        #print 'Creating a new Ticket with:'
        #print 'Ticket text:', ticket_text
        ticket_json = add_ticket(ticket_text, alarm_id, newToken, cookie, url)
        ticket = json.loads(ticket_json)
        print ticket['nid']
    elif param_close_ticket:
        ticket_id = args[0]
        #print 'Closing ticket with:'
        #print 'Ticket ID:', ticket_id
        close_ticket(ticket_id, newToken, cookie, url)
    elif param_comment_alarm or param_comment_ticket:
        alarm_id = args[0]
        comment_text = args[1]
        comment_node(alarm_id, comment_text, newToken, cookie, url)

if __name__ == '__main__':
    try:
        main()
    except Exception as e:
        print 'Error:', e
        sys.exit(1)
```

Script: HttpServerAvailability.sh

Ejemplo de parámetros: `"ERROR HTTP From: 192.168.72.105 To: 192.168.72.110"`

Salida en caso de éxito: `HttpServerAvailability: Ok`

Salida en caso de error: `HttpServerAvailability: Error`

Descripción: Obtiene información acerca de el estado del servicio *Http* en un elemento de red destino "*To*" y devuelve el resultado.

Código Bash Script:

```
#!/bin/bash
IP_FROM=`echo $* | grep -o "From: [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
IP_FROM=${IP_FROM:6}
IP_TO=`echo $* | grep -o "To: [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
IP_TO=${IP_TO:4}

wget -q -O /dev/null $IP_TO
if(test $? -eq 0) then
          echo "HttpServerAvailability: Ok"
else
          echo "HttpServerAvailability: Error"
fi
```

Script: pingChecker.sh

Ejemplo de parámetros: `"ERROR PING From: 192.168.72.105 To: 192.168.72.110"`

Salida en caso de éxito: `From: Ok (IP 192.168.72.105) From: Ok (IP 192.168.72.110)`

Salida en caso de error:
```
From: Error (IP 192.168.72.105) From: Error (IP 192.168.72.110)
From: Error (IP 192.168.72.105) From: Ok (IP 192.168.72.110)
From: Ok (IP 192.168.72.105) From: Error (IP 192.168.72.110)
```

Descripción: Verifica si se tiene acceso a través de a red por la *ip* a los elementos que conforman al mensaje de falla, tanto al emisor del mensaje como al elemento en problema y devuelve el resultado.

Código Bash Script:

```
#!/bin/bash
IP_FROM=`echo $* | grep -o "From: [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
IP_FROM=${IP_FROM:6}
IP_TO=`echo $* | grep -o "To: [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
IP_TO=${IP_TO:4}

WAIT=3
OUTPUT=""
ping -q -w $WAIT $IP_FROM >> /dev/null
if(test $? -eq 0) then OUTPUT="From: Ok (IP ${IP_FROM//./ })"; else OUTPUT="From: Error (IP ${IP_FROM//./ })"; fi
ping -q -w $WAIT $IP_TO >> /dev/null
if(test $? -eq 0) then OUTPUT=$OUTPUT" To: Ok (IP ${IP_TO//./ })"; else OUTPUT=$OUTPUT" To: Error (IP ${IP_TO//./ })"; fi
echo $OUTPUT
```

Script: sshDeleteTempFiles.sh

Ejemplo de parámetros: `"WARNING HDD From: 192.168.72.105 To: 192.168.72.110 96354k < 150434k on /dev/sda1"`

Salida en caso de éxito: `DeleteTempFiles: Ok`

Salida en caso de error: `DeleteTempFiles: Error`

Descripción: Elimina los archivos temporales del equipo destino, utilizando los datos proporcionados en el archivo de configuración de elementos de red *NES.TXT* y devuelve el estado de la operación.

Código Bash Script:

```
#!/bin/bash
IP_FROM=`echo $* | grep -o "From: [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
IP_FROM=${IP_FROM:6}
IP_TO=`echo $* | grep -o "To: [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
IP_TO=${IP_TO:4}

NE=(`cat ./pruebas/scripts/NES.TXT | while read line; do if(test \`echo $line | cut -d\  -f6\` == $IP_TO) then echo "$line"; fi done`)
SELF_NETWORK_INTERFACE=${NE[0]}
SELF_IP=(`ifconfig $SELF_NETWORK_INTERFACE | grep "inet:" | grep -o "[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`)
SELF_IP=${SELF_IP[0]}
NEM_IP_PORT="${NE[1]} ${NE[2]}"
PASSWORD=${NE[3]}
USER=${NE[4]}
URL=${NE[5]}
RAM_MIN=${NE[6]}
HDD=${NE[7]}
HDD_MIN=${NE[8]}
NETWORK_INTERFACE=${NE[9]}
NETWORK_MAX=${NE[10]}

SSH_OPTIONS=" -q -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no"
TO_DO="rm -f /home/$USER/*.tmp"
VALUE=`sshpass -p $PASSWORD ssh$SSH_OPTIONS $USER@$IP_TO "$TO_DO" 2> /dev/null`
if(test $? -eq 0) then
         echo "DeleteTempFiles: Ok"
else
         echo "DeleteTempFiles: Error"
fi
```

Script: sshHddFreeSpace.sh

Ejemplo de parámetros: `"WARNING HDD From: 192.168.72.105 To: 192.168.72.110 96354k < 150434k on /dev/sda1"`

Salida en caso de éxito: `HDD: Ok 202210k < 234645k on /dev/sda1`

Salida en caso de error: `HDD: Warning 201352k < 234645k on /dev/sda1`

Descripción: Obtiene información acerca del espacio libre en disco duro en un elemento de red utilizando los datos proporcionados en el archivo de configuración de elementos de red *NES.TXT* y advierte si el espacio libre es menor al mínimo requerido.

Código Bash Script:

```
#!/bin/bash
IP_FROM=`echo $* | grep -o "From: [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
IP_FROM=${IP_FROM:6}
IP_TO=`echo $* | grep -o "To: [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
IP_TO=${IP_TO:4}

NE=(`cat ./pruebas/scripts/NES.TXT | while read line; do if(test \`echo $line | cut -d\  -f6\` == $IP_TO) then echo "$line"; fi done`)
SELF_NETWORK_INTERFACE=${NE[0]}
SELF_IP=(`ifconfig $SELF_NETWORK_INTERFACE | grep "inet:" | grep -o "[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`)
SELF_IP=${SELF_IP[0]}
NEM_IP_PORT="${NE[1]} ${NE[2]}"
PASSWORD=${NE[3]}
USER=${NE[4]}
URL=${NE[5]}
RAM_MIN=${NE[6]}
HDD=${NE[7]}
HDD_MIN=${NE[8]}
NETWORK_INTERFACE=${NE[9]}
NETWORK_MAX=${NE[10]}
```

```
SSH_OPTIONS=" -q -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no"
TO_DO="VALUE=(\`df -BK | grep $HDD\`); echo \${VALUE[3]//K/}"
VALUE=`sshpass -p $PASSWORD ssh$SSH_OPTIONS $USER@$IP_TO "$TO_DO" 2> /dev/null`
if(test $VALUE -lt $HDD_MIN) then
         echo "HDD: Warning $VALUE""k < $HDD_MIN""k on $HDD"
else
         echo "HDD: Ok $VALUE""k < $HDD_MIN""k on $HDD"
fi
```

Script: sshNetworkTraffic.sh

Ejemplo de parámetros: `"WARNING NETWORK TRAFFIC From: 192.168.72.105 To: 192.168.72.110 150434b < 96354b on /dev/sda1"`

Salida en caso de éxito: `NetworkTraffic: Ok 43543b < 328410b on eth0`

Salida en caso de error: `NetworkTraffic: Warning 741287b > 328410b on eth0`

Descripción: Obtiene información acerca de el tráfico entrante de la red en un elemento de red utilizando los datos proporcionados en el archivo de configuración de elementos de red *NES.TXT* y advierte si el tráfico entrante por segundo es mayor al máximo establecido.

Código Bash Script:

```
#!/bin/bash
IP_FROM=`echo $* | grep -o "From: [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
IP_FROM=${IP_FROM:6}
IP_TO=`echo $* | grep -o "To: [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
IP_TO=${IP_TO:4}

NE=(`cat ./pruebas/scripts/NES.TXT | while read line; do if(test \`echo $line | cut -d\  -f6\` == $IP_TO) then echo "$line"; fi done`)
SELF_NETWORK_INTERFACE=${NE[0]}
SELF_IP=(`ifconfig $SELF_NETWORK_INTERFACE | grep "inet:" | grep -o "[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`)
SELF_IP=${SELF_IP[0]}
NEM_IP_PORT="${NE[1]} ${NE[2]}"
PASSWORD=${NE[3]}
USER=${NE[4]}
URL=${NE[5]}
RAM_MIN=${NE[6]}
HDD=${NE[7]}
HDD_MIN=${NE[8]}
NETWORK_INTERFACE=${NE[9]}
NETWORK_MAX=${NE[10]}

SSH_OPTIONS=" -q -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no"
TO_DO="FIELDS=\`cat /proc/net/dev | grep \"$NETWORK_INTERFACE\"\`"
TO_DO=$TO_DO";FIELDS=(\`echo \${FIELDS//\"$NETWORK_INTERFACE\"/} | grep -o \"[0-9]*\"\`)"
TO_DO=$TO_DO";PAST=\`echo \${FIELDS[0]}\`"
TO_DO=$TO_DO";sleep 1"
TO_DO=$TO_DO";FIELDS=\`cat /proc/net/dev | grep \"$NETWORK_INTERFACE\"\`"
TO_DO=$TO_DO";FIELDS=(\`echo \${FIELDS//\"$NETWORK_INTERFACE\"/} | grep -o \"[0-9]*\"\`)"
TO_DO=$TO_DO";NOW=\`echo \${FIELDS[0]}\`"
TO_DO=$TO_DO";expr \$NOW - \$PAST"
VALUE=`sshpass -p $PASSWORD ssh$SSH_OPTIONS $USER@$URL "$TO_DO" 2> /dev/null`
if(test $VALUE -lt $NETWORK_MAX) then
         echo "NetworkTraffic: Ok $VALUE""b < $NETWORK_MAX""b on $NETWORK_INTERFACE"
else
         echo "NetworkTraffic: Warning $VALUE""b > $NETWORK_MAX""b on $NETWORK_INTERFACE"
fi
```

Script: sshRamFreeSpace.sh

Ejemplo de parámetros: `"WARNING RAM From: 192.168.72.105 To: 192.168.72.110 150434k < 96354k"`

Salida en caso de éxito: `RamFreeSpace: Ok 238323k > 150320k`

Salida en caso de error: `RamFreeSpace: Warning 53935k < 150320k`

Descripción: Obtiene información acerca de la memoria *RAM* disponible en un elemento de red utilizando los datos proporcionados en el archivo de configuración de elementos de red *NES.TXT* y advierte si el espacio disponible es menor al mínimo requerido.

Código Bash Script:

```
#!/bin/bash
IP_FROM=`echo $* | grep -o "From: [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
IP_FROM=${IP_FROM:6}
```

```
IP_TO=`echo $* | grep -o "To: [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
IP_TO=${IP_TO:4}

NE=(`cat ./pruebas/scripts/NES.TXT | while read line; do if(test \`echo $line | cut -d\  -f6\` == $IP_TO) then echo "$line"; fi done`)
SELF_NETWORK_INTERFACE=${NE[0]}
SELF_IP=(`ifconfig $SELF_NETWORK_INTERFACE | grep "inet:" | grep -o "[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`)
SELF_IP=${SELF_IP[0]}
NEM_IP_PORT="${NE[1]} ${NE[2]}"
PASSWORD=${NE[3]}
USER=${NE[4]}
URL=${NE[5]}
RAM_MIN=${NE[6]}
HDD=${NE[7]}
HDD_MIN=${NE[8]}
NETWORK_INTERFACE=${NE[9]}
NETWORK_MAX=${NE[10]}

SSH_OPTIONS=" -q -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no"
TO_DO="VALUE=(\`free -k | grep \"Mem\"\`); echo \${VALUE[3]}"
VALUE=`sshpass -p $PASSWORD ssh$SSH_OPTIONS $USER@$URL "$TO_DO" 2> /dev/null`
if(test $VALUE -gt $RAM_MIN) then
        echo "RamFreeSpace: Ok $VALUE""k > $RAM_MIN""k"
else
        echo "RamFreeSpace: Warning $VALUE""k < $RAM_MIN""k"
fi
```

Script: sshRestartApache.sh

Ejemplo de parámetros: `"ERROR HTTP From: 192.168.72.105 To: 192.168.72.110"`

Salida en caso de éxito: `HttpServerAvailability: Ok`

Salida en caso de error: `HttpServerAvailability: Error`

Descripción: Reinicia el servicio *Http* (Apache) en un elemento de la red utilizando los datos proporcionados en el archivo de configuración de elementos de red *NES.TXT* y advierte el estado de la operación.

Código Bash Script:

```
#!/bin/bash
IP_FROM=`echo $* | grep -o "From: [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
IP_FROM=${IP_FROM:6}
IP_TO=`echo $* | grep -o "To: [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
IP_TO=${IP_TO:4}

NE=(`cat ./pruebas/scripts/NES.TXT | while read line; do if(test \`echo $line | cut -d\  -f6\` == $IP_TO) then echo "$line"; fi done`)
SELF_NETWORK_INTERFACE=${NE[0]}
SELF_IP=(`ifconfig $SELF_NETWORK_INTERFACE | grep "inet:" | grep -o "[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`)
SELF_IP=${SELF_IP[0]}
NEM_IP_PORT="${NE[1]} ${NE[2]}"
PASSWORD=${NE[3]}
USER=${NE[4]}
URL=${NE[5]}
RAM_MIN=${NE[6]}
HDD=${NE[7]}
HDD_MIN=${NE[8]}
NETWORK_INTERFACE=${NE[9]}
NETWORK_MAX=${NE[10]}

SSH_OPTIONS=" -q -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no"
TO_DO="echo $PASSWORD | sudo -S /etc/init.d/apache2 restart"
VALUE=`sshpass -p $PASSWORD ssh$SSH_OPTIONS $USER@$IP_TO "$TO_DO" 2> /dev/null`
if(test $? -eq 0) then
        echo "HttpServerAvailability: Ok"
else
        echo "HttpServerAvailability: Error"
fi
```

Script: monitor.sh

Ejemplo de parámetros: `No recibe parámetros.`

Salida en caso de éxito:

```
Tests for 192.168.72.111
Ok     : Ping
Ok     : Http
Ok     : RAM 164776k > 150000k on /dev/sda5
Ok     : HDD 360604k > 300000k on /dev/sda2
Ok     : Network traffic 212b < 1000b
```

Salida en caso de error:

```
Tests for 192.168.72.110
Error  : Ping

Tests for 192.168.72.120
Ok     : Ping
Error  : Http
Warning: RAM 51048k < 150000k on /dev/sda2
Warning: HDD 45936544k < 79320000k on /dev/sda5
Warning: Network traffic 7682b > 1000b
```

Descripción:

Es un monitor de elementos de red, que lee los atributos de cada elemento a partir de un archivo llamado *NES.TXT*. Verifica a cada segundo que haya conexión hacia el elemento a través de la ejecución del comando *ping*, que el servicio de apache esté levantado a través del comando *wget* y a través de conexiones vía *Secure Shell* solicita información del elemento de red, para conocer el espacio libre en *RAM*, disco duro y el flujo entrante de red. Si alguna de estas pruebas falla envía un mensaje de falla al gestor de fallas.

Código Bash Script:

```
#!/bin/bash
TRAP_IP_PORT="192.168.11.40 162"
SSH_OPTIONS=" -q -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no"
while [ 1 ]; do
         N=0
         while read LINE; do
                  LINES[$N]=$LINE
                  N=`expr $N + 1`
         done < <(cat "NES.TXT")
         I=1
         while(test $I -lt $N) do
                  PARAMETERS=(${LINES[$I]})
                  SELF_NETWORK_INTERFACE=${PARAMETERS[0]}
                  SELF_IP=(`ifconfig $SELF_NETWORK_INTERFACE |
                            grep "inet:" |
                            grep -o "[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`)
                  SELF_IP=${SELF_IP[0]}
                  NEM_IP_PORT="${PARAMETERS[1]} ${PARAMETERS[2]}"
                  PASSWORD=${PARAMETERS[3]}
                  USER=${PARAMETERS[4]}
                  URL=${PARAMETERS[5]}
                  RAM_MIN=${PARAMETERS[6]}
                  HDD=${PARAMETERS[7]}
                  HDD_MIN=${PARAMETERS[8]}
                  NETWORK_INTERFACE=${PARAMETERS[9]}
                  NETWORK_MAX=${PARAMETERS[10]}
                  echo "Tests for $URL"
                  #PING
                  ping -q -w 3 $URL >> /dev/null
                  if(test $? -eq 0) then
                            echo "Ok     : Ping"
                            #HTTP
                            wget -T 3 -q -O /dev/null $URL
                            if(test $? -eq 0) then
                                     echo "Ok     : Http"
                            else
                                     echo "Error  : Http"
                                     echo "ERROR HTTP From: $SELF_IP To: $URL" |
                                              nc -w 3 $NEM_IP_PORT >> /dev/null
                            fi
                            #RAM
                            DATA=(`sshpass -p $PASSWORD ssh$SSH_OPTIONS $USER@$URL
                                "free -k 2> /dev/null | grep Mem"`)
                            if(test $? -eq 0) then
                                     FREE=${DATA[3]}
                                     if(test $FREE -lt $RAM_MIN) then
                                              echo "Warning: RAM $FREE""k < $RAM_MIN""k on $HDD"
                                              echo "WARNING RAM From: $SELF_IP To: $URL $FREE""k < $RAM_MIN"
                                                       "k" | nc -w 3 $NEM_IP_PORT >> /dev/null
                                     else
                                              echo "Ok     : RAM $FREE""k > $RAM_MIN""k on $HDD"
                                     fi
                            fi
                            #HDD
                            DATA=(`sshpass -p $PASSWORD ssh$SSH_OPTIONS $USER@$URL
                                     "df -BK 2> /dev/null | grep $HDD"`)
                            if(test $? -eq 0) then
                                     FREE=${DATA[3]//K/}
                                     if(test $FREE -lt $HDD_MIN) then
                                              echo "Warning: HDD $FREE""k < $HDD_MIN""k on $HDD"
                                              echo "WARNING HDD From: $SELF_IP To: $URL $FREE""k < $HDD_MIN"
                                                       "k on $HDD" | nc -w 3 $NEM_IP_PORT >> /dev/null
                                     else
                                              echo "Ok     : HDD $FREE""k > $HDD_MIN""k on $HDD"
                                     fi
                            fi
                            #NETWORK
                            TEST="FIELDS=\`cat /proc/net/dev | grep \"$NETWORK_INTERFACE\"\`"
                            TEST=$TEST";FIELDS=(\`echo \${FIELDS//\"$NETWORK_INTERFACE\"/}
```

```
                                        | grep -o \"[0-9]*\"\`)"
                                TEST=$TEST";PAST=\`echo \${FIELDS[0]}\`"
                                TEST=$TEST";sleep 1"
                                TEST=$TEST";FIELDS=\`cat /proc/net/dev | grep \"$NETWORK_INTERFACE\"\`"
                                TEST=$TEST";FIELDS=(\`echo \${FIELDS//\"$NETWORK_INTERFACE\"/}
                                        | grep -o \"[0-9]*\"\`)"
                                TEST=$TEST";NOW=\`echo \${FIELDS[0]}\`"
                                TEST=$TEST";expr \$NOW - \$PAST"
                                DATA=`sshpass -p $PASSWORD ssh$SSH_OPTIONS $USER@$URL "$TEST"
                                        2> /dev/null`
                                if(test $? -eq 0) then
                                        if(test $DATA -gt $NETWORK_MAX) then
                                                echo "Warning: Network traffic $DATA""b > $NETWORK_MAX""b"
                                                echo "WARNING NETWORK TRAFFIC From: $SELF_IP To: $URL $DATA"
                                                        "b > $NETWORK_MAX""b" | nc -w 3 $NEM_IP_PORT >> /dev/null
                                        else
                                                echo "Ok     : Network traffic $DATA""b < $NETWORK_MAX""b"
                                        fi
                                fi
                        else
                                echo "Error  : Ping"
                                echo "ERROR PING From: $SELF_IP To: $URL" | nc -w 3 $NEM_IP_PORT
                                        >> /dev/null
                        fi
                        I=`expr $I + 1`
                        echo
                done
                sleep 1
done
```

Nombre de Archivo: NES.TXT

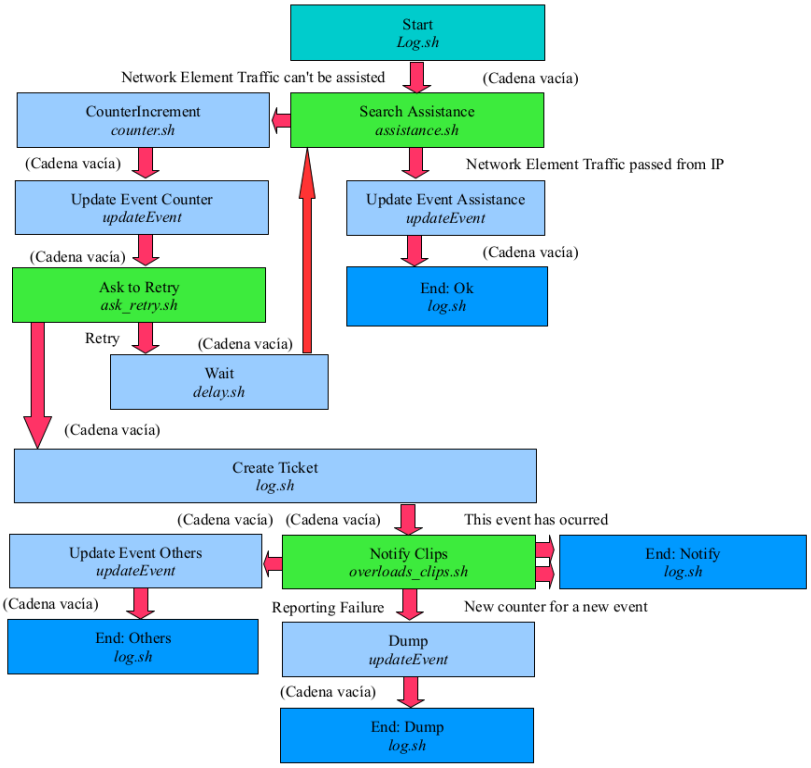
Descripción: Archivo de configuración que describe los siguientes campos para cada elemento de la red: La tarjeta de red del monitor por la que se comunicará al elemento de la red, la *ip* del monitor en esa tarjeta, el puerto por el cual enviará mensajes de falla al gestor de red, la clave para la conexión *ssh*, el usuario para la conexión *ssh*, la *ip* a conectarse vía *ssh*, la cantidad de memoria *RAM* en kilobytes mínima disponible en el elemento antes de enviar un mensaje de advertencia, la ubicación del disco duro a observar, la cantidad mínima de espacio libre disponible en kilobytes en el disco duro, la tarjeta de red a la cual observará el tráfico de red entrante y por último la cantidad máxima de tráfico entrante en bytes.

```
<TRedMonitors.sh> <IpNEM> <PuertoNEM> <PswNE> <UsrNE> <IpNE> <MinRAMks> <PathHDD> <MinHDDks> <Tar.RedNE> <MaxNetTrafficB>
eth0 127.0.0.1 8003 djtf dinamica 192.168.72.110 150000 /dev/sda1 98000000 eth0 1000
eth0 127.0.0.1 8003 sfed dinamica 192.168.72.111 150000 /dev/sda2 300000 eth1 1000
eth0 127.0.0.1 8003 jyrv dinamica 192.168.72.120 150000 /dev/sda5 79320000 eth0 1000
```

Procedimientos utilizados en las pruebas de correlación de eventos

Procedimiento:	Overloads
Condición:	`Overload Event`
Ejemplo de mensaje:	`long.colaTX:1000 192.167.6.121 Overload Event Bytes RX:453917 (453.9 KB) TX bytes:453917 (453.9 KB)`
Descripción:	Este procedimiento tiene como condición a un evento de sobrecarga. Lo que hace es intentar desviar el tráfico a algún otro elemento de red, hará este intento 3 veces y en caso de no recibir ayuda notificará a la instancia de *CLIPS*. En caso de que *CLIPS* detecte que hay una gran cantidad de fallas con respecto a la misma *ip*, *CLIPS* responderá con un reporte de las ip's relacionadas a la misma *ip* hacia un *log* de eventos.

Diagrama:



Código XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<finiteStateMachine name="Overloads" initialState="Start" condition="Overload Event">
          <states>
                    <state name="Start" timeOut="10000"
                              command="bash ./src/log.sh logs/FAULTS_START.LOG @" />
                    <state name="Search Assistance" timeOut="80000"
                              command="bash ./src/assistance.sh @" />
                    <state name="Update Event Assistance" timeOut="10000"
                              command="updateEvent" />
                    <state name="End: Ok" timeOut="10000"
                              command="bash ./src/log.sh logs/FAULTS_END.LOG @" />
                    <state name="Counter Increment" timeOut="10000"
                              command="bash ./src/counter.sh RetryCounter @" />
                    <state name="Update Event Counter" timeOut="10000"
                              command="updateEvent" />
                    <state name="Ask to Retry" timeOut="10000"
                              command="bash ./src/ask_retry.sh RetryCounter 3 @" />
                    <state name="Wait" timeOut="100000"
                              command="bash ./src/delay.sh 3" />
                    <state name="Create Ticket" timeOut="10000"
                              command="bash ./src/log.sh logs/FAULTS_TT.LOG @" />
                    <state name="Notify Clips" timeOut="10000"
```

```
                    command="bash ./src/overloads_clips.sh &amp;quot;@&amp;quot;" />
          <state name="End: Notify" timeOut="10000"
                    command="bash ./src/log.sh logs/FAULTS_CLIPS_NOTIFIED.LOG @" />
          <state name="Dump" timeOut="10000"
                    command="updateEvent" />
          <state name="End: Dump" timeOut="10000"
                    command="bash ./src/log.sh logs/FAULTS_CLIPS_DUMP.LOG @" />
          <state name="Update Event Others" timeOut="10000"
                    command="updateEvent" />
          <state name="End: Others" timeOut="10000"
                    command="bash ./src/log.sh logs/FAULTS_CLIPS_OTHERS.LOG @" />
     </states>
     <transitions>
          <transition from="Start" to="Search Assistance"
                    condition="" />
          <transition from="Search Assistance" to="Update Event Assistance"
                    condition="Network Element Traffic passed from IP" />
          <transition from="Update Event Assistance" to="End: Ok"
                    condition="" />
          <transition from="Search Assistance" to="Counter Increment"
                    condition="Network Element Traffic can't be assisted" />
          <transition from="Counter Increment" to="Update Event Counter"
                    condition="" />
          <transition from="Update Event Counter" to="Ask to Retry"
                    condition="" />
          <transition from="Ask to Retry" to="Wait"
                    condition="Retry" />
          <transition from="Wait" to="Search Assistance"
                    condition="" />
          <transition from="Ask to Retry" to="Create Ticket"
                    condition="" />
          <transition from="Create Ticket" to="Notify Clips"
                    condition="" />
          <transition from="Notify Clips" to="Dump"
                    condition="Reporting Failture" />
          <transition from="Dump" to="End: Dump"
                    condition="" />
          <transition from="Notify Clips" to="End: Notify"
                    condition="This event has occurred" />
          <transition from="Notify Clips" to="End: Notify"
                    condition="New counter for a new event" />
          <transition from="Notify Clips" to="Update Event Others"
                    condition="" />
          <transition from="Update Event Others" to="End: Others"
                    condition="" />
     </transitions>
</finiteStateMachine>
```

Scripts utilizados en las pruebas de correlación de eventos

Script: ask_retry.sh

Ejemplo de parámetros: `"Counter Name" max "Event: Counter Name 10 / More ..."`

Salida en caso de éxito: `Retry`

Salida en caso de error: `End`

Descripción: Busca un contador dentro de una cadena de texto y si la cuenta excede al valor máximo establecido en los parámetros la salida del *script* es *End* en caso contrario la salida es *Retry*.

Código Bash Script:

```
#!/bin/bash
#./ask_retry.sh "Counter Name" max "Event: Counter Name 10 / More ..."
field=`echo "$*" | grep -o "|$1 [0-9]\+|"`
if(test ${#field} -gt 0) then
     value=`echo "$field" | grep -o "[0-9]\+"`
     if(test $value -lt $2) then
          echo "Retry"
     else
          echo "End"
     fi
else
     echo "Error"
fi
```

Script: assistance.sh

Ejemplo de parámetros: `"Counter Name" max "Event: Counter Name 10 / More ..."`

Salida en caso de éxito: `"Event: IP 123.123.123.123 | Network Element Traffic passed from IP: 123.123.123.123 to IP: 3.7.1.9"`

Salida en caso de error: `"Event: IP 123.123.123.123 | Network Element Traffic can't be assisted from IP: 123.123.123.123 all Network Elements Busy"`

Descripción: Simula hacer la transferencia de tráfico asignada a una dirección *ip* a otra, con dos posibles resultados, en el primero la transferencia es exitosa y en la segunda se avisa que la transferencia no pudo ser realizada.

Código Bash Script:

```
#!/bin/bash
ip=`echo $* | grep -o "[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
sleep $((RANDOM%5))
if(test $((RANDOM%10)) -lt 2) then
         echo "$* | Network Element Traffic passed from IP: $ip to IP: $((RANDOM%255)).$((RANDOM%255)).$((RANDOM%255)).$((RANDOM%255))"
else
         echo "$* | Network Element Traffic can't be assisted from IP: $ip all Network Elements Busy"
fi
```

Script: counter.sh

Ejemplo de parámetros: `"Counter Name" max "Event: Counter Name 10 / More ..."`

Salida en caso de éxito: `"Event: More ... | Counter Name 11 |"`

Salida en caso de error: `"Event: More ... | Counter Name 1 |"`

Descripción: Agrega a un texto un contador identificado con el nombre que se pase como parámetro o en caso de ya existir, lo incrementa en uno.

Código Bash Script:

```
#!/bin/bash
#./counter.sh "Counter Name" "Event: Counter Name 10 / More ..."
field=`echo "$*" | grep -o "|$1 [0-9]\+|"`
if(test ${#field} -gt 0) then
         value=`echo "$field" | grep -o "[0-9]\+"`
         new_value=`expr $value + 1`
         new_string=`echo "$*" | sed -e "s/$field/\|$1 $new_value\|/g"`
else
         new_string=`echo "$*|$1 0|"`
fi
echo "${new_string:${#1}+1}"
```

Script: delay.sh

Ejemplo de parámetros: `10`

Salida en caso de éxito: `""`

Salida en caso de error: `""`

Descripción: Realiza una pausa de una cantidad de segundos definida en los parámetros.

Código Bash Script:

```
#!/bin/bash
sleep $1
```

Script: log.sh

Ejemplo de parámetros: `“carpeta/log.txt” “Mensaje de texto a agregar al log.”`

Salida en caso de éxito: `""`

Salida en caso de error: `""`

Descripción: Realiza una pausa de una cantidad de segundos definida en los parámetros.

Código Bash Script:

```
#!/bin/bash
string=$*
name=$1
length=${#name}
echo `date` "${string:length+1}" >> $1
```

Script: overloads_clips.sh

Ejemplo de parámetros: `“Evento: IP 123.123.123.123”`

Salida en caso de éxito: `(Se mantiene corriendo el programa)`

Salida en caso de error: `"Shell <verbose> <host> <port> <timeOut> [message]"`

Descripción: Inserta una *ip* dentro de la base de hecho en un servidor de *CLIPS*.

Código Bash Script:

```
#!/bin/bash
ip=`echo $* | grep -o "[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"`
java Shell 0 localhost 8004 10 "(assert (faultEvent (name \"Overload on $ip\")))\n(run)"
```

Sentencias utilizadas en las pruebas de correlación de eventos

Archivo: Overloads.clp

Descripción: Recibe información acerca de mensajes de falla provenientes de alguna *ip*, en caso de haber reportados más de 2 mensajes de falla de la misma *ip*, se despliega un reporte mostrando la *ip* y el número de veces que ha presentado el mensaje de falla, así como el contador de ese mensaje se reinicia a cero. En caso de ser la primera ves responderá avisando que ese mensaje es la primera vez que ocurre. En caso de no ser la primera ves ni una vez mayor a la segunda entonces sólo muestra el número de veces que se tiene conocimiento de ese mensaje de falla.

Código CLIPS:

```
(deftemplate counter "Counter" (slot n (type INTEGER)))
(deftemplate faultEvent "Fault Event" (slot name (type STRING)))
(deftemplate failture "Failture" (slot name (type STRING)) (slot times (type INTEGER)))
(deftemplate report "Report" (slot test (type INTEGER)))
(defrule faultEventNew "Fault Event New"
          ?c <- (counter (n ?n))
          ?f <- (faultEvent (name ?name))
   (not (failture (name ?name) (times ?times)))
          =>
          (modify ?c (n (+ ?n 1)))
          (assert (failture (name ?name) (times 1)))
          (retract ?f)
          (printout t "New counter for a new event." crlf))
(defrule faultEventCount "Fault Event Count"
          ?c <- (counter (n ?n))
          ?f <- (faultEvent (name ?name))
   ?g <- (failture (name ?name) (times ?times))
          =>
          (modify ?g (times (+ ?times 1)))
          (retract ?f)
          (printout t "This event has occurred " ?times " times." crlf))
(defrule reportSeriousFault "Report Serious Fault"
          ?c <- (counter (n ?n))
          ?r <- (report (test ?test))
          (test (and (= ?test 0) (> ?n 2)))
```

```
          =>
          (modify ?r (test 1))
          (printout t "Reporting " ?n " faults events!" crlf))
(defrule reporter "Reporter"
          ?c <- (counter (n ?n))
          ?r <- (report (test ?test))
          ?f <- (failture (name ?name) (times ?times))
          (test (and (= ?test 1) (> ?n 0)))
          =>
          (modify ?c (n (- ?n 1)))
          (retract ?f)
          (printout t "Reporting Failture: " ?name " " ?times " times." crlf))
(defrule stopReporting "Stop Reporting"
          ?r <- (report (test ?test))
          ?c <- (counter (n ?n))
          (test (and (= ?test 1) (= ?n 0)))
          =>
          (modify ?r (test 0))
          (printout t "Stop Reporting." crlf))
(deffacts default (counter (n 0)) (report (test 0)))
(reset)
;(assert (faultEvent (name "Torre 1")))
;(run)
```

Apéndice II

Glosario

Términos técnicos

Actor Es algo externo al software que intercambia información con el sistema, puede ser un usuario u otro sistema.

Actuadores Dispositivos capaces de transformar energía hidráulica, neumática o eléctrica en la activación de un proceso con la finalidad de generar un efecto sobre un proceso automatizado.

Administración remota Funcionalidad de algunos programas que permiten realizar ciertos tipos de acciones desde un equipo local y que las mismas se ejecuten en otro equipo remoto.

Arista Es la unión entre dos vértices.

ARP *Address Resolution Protocol* Protocolo de resolución de dirección.

Base de datos Conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

BGP *Border Gateway Protocol* Protocolo de interfaz de frontera.

Big Brother Sistema de Monitorización.

Cable coaxial Cable utilizado para transportar señales eléctricas de alta frecuencia que posee dos conductores concéntricos, uno central, llamado vivo, encargado de llevar la información, y uno exterior, de aspecto tubular, llamado malla o blindaje, que sirve como referencia de tierra y retorno de las corrientes.

Cadena de texto Es una secuencia ordenada de longitud arbitraria finita de elementos que pertenecen a un cierto lenguaje formal o alfabeto análogas a una frase o a una oración.

Compilador Un compilador es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar.

Caso de uso Define para qué procesos se usará un software.

Control de flujo Debe tener la capacidad de negar el envío de mensajes cuando el canal se encuentre saturado.

Correlación Define el nivel de relación entre dos eventos.

CPU *Central Processing Unit* Unidad Central de Procesamiento, procesador o microprocesador, es el componente principal del ordenador y otros dispositivos programables, que interpreta las instrucciones contenidas en los programas y procesa los datos.

DCN *Data Communication Network* Redes de comunicación de datos, transportan la información de gestión de una red.

Direccionamiento Es la capacidad de identificar direcciones para poder hacer llegar un mensaje a su destino.

Dominio público Implica que las obras pueden ser explotadas por cualquier persona, pero

	siempre respetando los derechos morales de sus autores, pues lo que en realidad expira son los derechos de autor de carácter patrimonial.
EC	*Element Controller* o Gestor de elementos *Element Manager*, El Controlador de elementos registra eventos relacionados con la administración de la red.
Enlaces	Son canales físicos de comunicación entre nodos o equipos.
Enmascaramiento de alarma	Previene el reporte de alarmas innecesarias al deshabilitarlas o habilitarlas.
Enrutamiento	Cada nodo debe buscar la ruta más rápida para el envío de cada mensaje, y para esto toma en cuenta la congestión de la red.
Equipos terminales	Equipos a través de los cuales se obtiene entrada a la red por medio de un canal de acceso.
Escalable	Propiedad deseable de un sistema, una red o un proceso, que indica su habilidad para reaccionar y adaptarse sin perder calidad.
Estadística	Ciencia formal que estudia la recolección, análisis e interpretación de datos de una muestra representativa, ya sea para ayudar en la toma de decisiones o para explicar condiciones regulares o irregulares de algún fenómeno o estudio aplicado, de ocurrencia en forma aleatoria o condicional.
Estática	Acumulación de un exceso de carga eléctrica en una zona con poca conductividad eléctrica, un aislante, de manera que la acumulación de carga persiste.
Ethereal/Wireshark	Analizador de protocolos utilizado para realizar análisis y solucionar problemas en redes de comunicaciones, para desarrollo de software y protocolos, y como una herramienta didáctica para educación.
Evento	Es una notificación dentro de la red.
Explosión combinatoria	Crecimiento exponencial del tamaño del espacio de búsqueda en relación a las variables y sus dominios.
Falla	Es un problema que ocurre en el hardware o en el software de la red, que pueden ser generadas por causas externas o internas.
Fibra óptica	Medio de transmisión empleado habitualmente en redes de datos. Consta de un hilo muy fino de material transparente, vidrio o materiales plásticos, por el que se envían pulsos de luz que representan los datos a transmitir.
FIFO	Del inglés *first in, first out*, guarda analogía con las personas que esperan en una cola y van siendo atendidas en el orden en que llegaron, es decir, que la primera persona que entra es la primera persona que sale.
Flujo de eventos alternativos o excepcionales	Acciones que ocurren en situaciones anormales.
Flujo de eventos normales	Describe el flujo ideal de acciones esperadas entre el actor y el sistema.
Formateo	Se necesita modificar la codificación del mensaje para que pueda transitar entre redes que manejan distintos protocolos.
GNU	El sistema GNU es un proyecto iniciado por Richard Stallman con el objetivo de crear un sistema operativo completamente libre.
GPS	Sistema de posicionamiento global por satélite que permite determinar en todo el mundo la posición de un objeto, una persona o un vehículo con

	una precisión habitual dada en metros.
Grafo	Es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto.
Hardware	Se refiere a todas las partes tangibles de un sistema informático y sus componentes son: eléctricos, electrónicos, electromecánicos y mecánicos.
Híbrido	Denotan a los sistemas que emplean, en paralelo, una combinación de modelos.
HP Openview	Sistema de Monitorización de Hewlett-Packard.
Infrarrojo	Tipo de radiación electromagnética y térmica de mayor longitud de onda que la luz visible, pero menor que la de las microondas.
Inteligencia	Capacidad que tiene una entidad con respecto a la toma de decisiones en torno a la satisfacción de un conjunto de propósitos, que implican la resolución total o parcial de una serie de problemas.
Interfaz	Se utiliza para nombrar a la conexión física y funcional entre dos sistemas o dispositivos de cualquier tipo dando una comunicación entre distintos niveles.
Internet	Conjunto descentralizado de redes de comunicación interconectadas que utilizan la familia de protocolos TCP/IP, garantizando que las redes físicas heterogéneas que la componen funcionen como una red lógica única, de alcance mundial.
Lenguaje natural	Es el lenguaje hablado o escrito por humanos para propósitos generales de comunicación.
LAN	*Local Area Network* Red de área local, enlazan computadoras instaladas en un edificio u oficina.
Línea de comandos	Es un método que permite a las personas dar instrucciones a algún programa informático por medio de una línea de texto simple.
MAN	*Metropolitan Area Network* Red de área metropolitana, se extiende dentro de una misma ciudad.
Masking	Previene el reporte de alarmas innecesarias al deshabilitarlas o habilitarlas.
MD	*Mediation Device* Dispositivos de mediación dentro de una red.
Nagios	Sistema de Monitorización de redes.
NE	*Network Element* Elementos de red.
NOC	*Network Operations Center* Centro de Operaciones de Red para gestionar la red.
Nodos	Son equipos que proveen enlaces físicos entre los canales de comunicación de la red.
Nombre de dominio	Su propósito principal es traducir las direcciones *ip* de cada nodo activo en la red, a términos memorizables y fáciles de encontrar.
OOB	*Out of Bounds* Fuera de los límites.
OS	*Operating Systems* Sistemas operativos.
OSS	Procesos de soporte para el mantenimiento del inventario de red, servicios de aprovisionamiento, configuración de los elementos de red y software para la gestión de fallas.
Paralelizable	Es un algoritmo que puede ser ejecutado por partes en el mismo instante de tiempo por varias unidades de procesamiento, para finalmente unir

	todas las partes y obtener el resultado correcto.
Patrones	Forma, modelo, simulación o paradigma que pueden ser usadas para crear o generar entidades o partes de una entidad, especialmente si las entidades generadas tienen lo suficiente en común como para que sea posible inferir o discernir el patrón, en cuyo caso se dice que exhiben el patrón.
Ping	Utilidad que diagnóstica en redes de computadoras que comprueba el estado de la conexión de un sistema hacia uno o varios equipos remotos de una red TCP/IP por medio del envío de paquetes ICMP de solicitud y de respuesta.
Poscondiciones	Define el estado del sistema después de la terminación exitosa de una tarea.
Precondiciones	Estado del sistema previo requerido para que se pueda realizar una tarea.
Programación imperativa	Paradigma de programación que describe la programación en términos del estado del programa y sentencias que cambian dicho estado.
Programación lógica	Paradigma de programación que gira en torno al concepto de predicado, o relación entre elementos.
Programación Orientada a Objetos	Paradigma de programación que usa los objetos en sus interacciones, para diseñar aplicaciones y programas informáticos.
Protocolo	Conjunto de reglas normalizadas para la representación, señalización, autenticación y detección de errores necesario para enviar información a través de un canal de comunicación.
Puertos de acceso	Es una interfaz para comunicarse con un programa a través de una red.
QA	*Q Adapter* Adaptadores Q, convierten datos de entrada y salida a un formato compatible con la red.
Red	Conjunto de equipos informáticos y software conectados entre sí por medio de dispositivos físicos que envían y reciben impulsos eléctricos, ondas electromagnéticas o cualquier otro medio para el transporte de datos, con la finalidad de compartir información, recursos y ofrecer servicios.
Red de cobertura urbana	Se compone de redes metropolitanas.
Red global de telecomunicaciones	Se compone de redes nacionales.
Red metropolitana	Se compone de redes de cobertura urbana.
Red nacional	Se compone de redes metropolitanas.
Red privada	Cuando se utiliza para uso personal, sin dar acceso a terceros.
Red pública	Se utiliza para ofrecer servicios de telecomunicaciones al público en general.
Repetición	Existen protocolos que detectan si ha habido algún error en la transmisión y si es así puede solicitar una retransmisión.
Semántica	Refiere a los aspectos del significado, sentido o interpretación de signos lingüísticos como símbolos, palabras, expresiones o representaciones formales.
Shell	Programas que proveen una interfaz de usuario para acceder a los servicios del sistema operativo.
Sistema inteligente	Programa de computación que reúne características y comportamientos asimilables al de la inteligencia humana o animal.

SNMP	*Simple Network Management Protocol* Protocolo simple de administración de red.
Socket	Concepto abstracto por el cual dos programas pueden intercambiar cualquier flujo de datos incluso si se encuentran en computadoras distintas, generalmente de manera fiable y ordenada a través de la red.
Software	Equipamiento lógico o soporte lógico de un sistema informático, comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos, que son llamados hardware.
SWAP	Del inglés "intercambiar". Espacio de intercambio que es una zona de la unidad de almacenamiento que se usa para guardar datos de los procesos que no han de mantenerse en memoria física.
Switches	Dispositivo digital lógico de interconexión de redes de computadoras que opera en la capa de enlace de datos del modelo OSI. Su función es interconectar dos o más segmentos de red, de manera similar a los puentes de red, pasando datos de un segmento a otro de acuerdo con la dirección MAC de destino de las tramas en la red.
Teorema de Bayes	Expresa la probabilidad condicional de un evento aleatorio A dado B en términos de la distribución de probabilidad condicional del evento B dado A y la distribución de probabilidad marginal de sólo A.
Ticketing System	Sistema de manejo de incidencias.
Topología	Determina la configuración de las conexiones entre sus elementos.
Traceroute	Es una consola de diagnóstico que permite seguir la pista de los paquetes que vienen desde un punto de red.
Transductor	Dispositivo capaz de transformar o convertir un determinado tipo de datos de entrada, en otros diferentes de salida.
Transmisión	Los mensajes se adaptan al canal, para su transporte eficiente y efectivo a través de la red.
Tupla	Secuencia ordenada de objetos, esto es, una lista con un número limitado de objetos.
Vértice	Es el punto donde se encuentran dos o más semirrectas que conforman un ángulo.
WAN	Wide Area Network Red de área amplia.
World Wide Web Consortium	*W3C* Es un consorcio internacional que produce recomendaciones para la *World Wide Web* Red de redes mundial o internet.
WS	*Workstations* Estaciones de trabajo, permiten a los operadores acceder a la información de gestión.
XML	Lenguaje de marcas desarrollado por el *W3C*. Permite definir la gramática de lenguajes específicos para estructurar documentos a través de redes.

Apéndice III

Bibliografía

Fuentes consultadas

1. Recomendación M.1130 del CCITT **Manenimiento: sistemas y servicios de telecomunicaciones móviles**.
2. S. Díaz, J. I. Escudero, J. Luque, **Expert system - Based alarm managment in communication networks**, Dpto. Tecnología Electrónica, Universidad de Sevilla, Spain. 2000.
3. Shu-Hsien Liao, **Expert system methodologies and applications - a decade review from 1995 to 2004**. Volume 28, Issue 1, January 2005, Pages 93–103. Tamkang University, Taiwan. 2005.
4. R. J. Patton, J. Chen, T. M. Siew, **Fault Diagnosis in Nonlinear dynamic systems via neural-networks**, Proc. IEE Int. Conference Control. 1994
5. M. Garijo, A. Cancer, J. J. Sánchez, **A Multiagent System for Cooperative Network-Fault Management**, ETSI Telecomunicación, Universidad Politécnica de Madrid, Dpto. de Ingeniería de Sistemas Telemáticos, Spain. 1996.
6. Mark T. Sutter, Paul E. Zeldin, **Designing Expert Systems for Real-Time Diagnosis of Self-Correcting Networks**, Network Equipment Technologies, Network Management Systems, USA. 1988.
7. Antonio M. Campos, Daniel F. García, **A Dynamic Scheduling Algorithm for Real-Time Expert Systems**, Department of Informatics, University of Oviedo, Campus de Viesques, Spain. 2002.
8. Y. Pencolé, M-O Cordier, L. Rozé, **A decentralized model-based diagnostic tool for complex systems**. IRISA, University of Rennes I, campus Beaulieu, France. IRISA, INSA, campus Beaulieu, France. 2001.
9. M. El-Darieby, A. Bieszczad, **Intelligent Mobile Agents: Towards Network Fault Management Automation**, Carleton University, Ottawa, Canada. Bell Laboratories, Lucent Technologies, USA. 1999.
10. Yannick Pencolé, Marie-Odile Cordier, **A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks**, CSL, The Australian National University, Australia. IRISA/Université de Rennes, France. 2005.
11. R. Sterritt, K. Adamson, C.M. Shapcott, E.P. Curran, **Data Mining telecommunications network data for fault management and development testing. Faculty of Informatics**, University of Ulster, Northern Ireland. 2000.
12. Mouhammd Al-Kasassbeh, Mo. Adda, **Network fault detection with Wiener filter-based agent, Journal of Network and Computer Applications**, Volume 32, Issue 4, Pages 824–833. 2009.
13. Braun, J.E., **Intelligent Building Systems - Past, Present, and Future, American Control Conference**, ACC '07. Page(s): 4374 - 4381. 2007.
14. Michael Negnevitsky. **A Guide to Intelligent Systems. Second Edition**. Pearson Education Limited, England. 2005.
15. Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, Marc Tommasi. **Tree Automata Techniques and Applications**. University of Lille 1, France. 2008.
16. Frank Neven, **Automata, Logic, and XML**. University of Limburg, London. 2002.
17. Chris Culbert, **CLIPS Reference Manual - Basic Programming Guide Volume I Version 6.24**, Space

Center, USA. 2006.
18. Chris Culbert, **CLIPS Reference Manual - Advanced Programming Guide Volume II Version 6.24**, Space Center, USA. 2006.
19. Viviana Espinosa Rojas, Gina Paola Pulido Uriza, **Sistema experto EoS: Gestión de fallas con diagnóstico experto para turbocompresores utilizados en la extracción de petróleo**, Pontificia Universidad Javeriana, Bogotá. 2004.
20. Carlos Vicente, **Introducción a la Gestión de Redes**, Universidad de Oregon, USA. 2011.
21. Katalin M. Hangos, Rozália Lakner, Miklós Gerzson, **Intelligent Control Systems An Introduction with Examples**, Computer and Automation Research Institute of the Hungarian Academy of Sciences, University of Veszprém, Hungary. 2002.
22. Dolores Molina González, **Gestión de redes de telecomunicaciones**, Dpto. de Ingeniería Electrónica, de Telecomunicación y Automática, Universidad de Jaén, España. 2011.
23. Erik T. Ray, **Learning XML, First Edition**, O'Reilly Media, USA. 2001.
24. Klaus Krippendorff, **Combinatorial Explosion**, Web Dictionary of Cybernetics and Systems, USA. 2010.
25. Raúl Costa, Nuno Cachulo, Paulo Cortez, **An Intelligent Alarm Management System for Large-Scale Telecommunication Companies**, Department of Information Systems/Algoritmi R&D Centre, University of Minho. Portugal. 2009.
26. Renee Keffer, Denise W. Gürer, Irfan Khan, Richard Ogier, **An Artificial Intelligence Approach to Network Fault Management**, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025 USA, Sprint, 9300 Metcalf, Overland Park, KS 66212 USA. 1996.
27. S. Díaz, J. I. Escudero, J. Luque, **Expert System Based Alarm Management in Communication Networks**, Dpto. Tecnología Electrónica. Universidad de Sevilla. Avda. Reina Mercedes, s/n. 41012 Sevilla, Spain. 2000.
28. J. Luque1, F. Gonzalo2, J. I. Escudero1, A. Carrasco1, **TMN Versus SNMP-Based Network Management Systems**, Universidad de Sevilla, Endesa Ingeniería de Telecomunicaciones, Enditel, Spain. 1999.
29. Joaquín Luque Rodríguez, José I. Escudero, Francisco Pérez García, Carlos León de Mora, Sergio Martín Guillén, Manuel Mejías Risoto, Sergio Díaz Ruíz, Ma, Carmen Romero Ternero. **Gestión experta de fallos en redes de comunicaciones**. Departamento de Tecnología Electrónica, Universidad de Sevilla, España. 2000.
30. Douglas W. Stevenson. **NMS: Network Management White Paper** http://www.sce.carleton.ca/netmanage/NetMngmnt/NetMngmnt.html#MOM (1/27/2013)
31. Booch G., Rumbaugh J., Jacobson I. **The Unified Modeling Languaje. Users guide**, Addison Wesley 1999.