UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

MÁQUINA DE INFERENCIA EN TIEMPO REAL PARA UN SISTEMA EXPERTO DE TELECOMUNICACIONES

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:

DANIEL TREJO BAÑOS

DIRECTOR DE LA TESIS: DR. NICOLÁS KEMPER VALVERDE

MÉXICO, D.F. **2011.**

Dedicatoria

Es el final de un largo camino, y espero que el principio de uno más interesante. No habría llegado a este punto de no ser por el apoyo incondicional de mis padres Tomás y María de Lourdes y de mi hermano Alejandro, no puedo expresar con palabras mi agradecimiento y mi amor por ustedes.

A mis amigos, perdón por no mencionarlos a todos de nombre, y decirles a cada uno lo que significan para mí, pero afortunadamente son muchos y desafortunadamente el espacio es poco. Pero a todos y cada uno de ustedes les agradezco su compañía y el hacerme mejor persona día a día.

A mí tutor NicolásKemper agradezco su guía y consejos durante estos últimos años.

A mis sinodales, profesores, colegas y compañeros de laboratorio, les agradezco que hayan compartido sus conocimientos y puntos de vista conmigo, parte de ustedes se encuentra en este trabajo.

Daniel Trejo Baños

Contenido

1 Introducción

1.1. Definición del problema

Una red de gestión de telecomunicaciones (Telecommunications Management Network: TMN) presenta un modelo orientado a objetos, definido por un número de estándares y basado sobre el modelo de comunicaciones de siete capas OSI. Es una red conceptualmente separada que interactúa con la red de telecomunicaciones en diferentes puntos (1).

Los Sistemas de Gestión de Red TMN están concebidos para ser capaces de gestionar:

- Redes telefónicas
- Redes LAN yWAN
- Redes ISDN
- Redes de Servicios Móviles
- Servicios de Red Inteligente y de Valor Agregado
- Redes Digitales avanzadas de Banda Ancha, tales como:
- Redes SONET/SDH
- Redes ATM
- Redes B-ISDN

La arquitectura funcional del estándar TMN (Telecommunications Management Network) está basada en los siguientes de bloques de función TMN (1):

- Bloque de función de sistemas de operaciones (Operations Systems Function: OSF)
- Bloque de función de elemento de red (Network Element Function: NEF)
- Bloque de función de estación de trabajo (Worstation Function: WSF)
- Bloque de función de mediación (Mediation Function: MF)
- Bloque de función de adaptador Q (Q Adaptaor Function: QAF)

De acuerdo a la terminología TMN, las OSF de la gestión de red están separadas en cuatro capas jerárquicas. Cada capa de la jerarquía define un grupo apropiado de operaciones de gestión. Estas capas son construidas una sobre otra; ellas (y sus operaciones apropiadas) están muy interrelacionadas.

El estándar TMN define las siguientes cuatro capas de la OSF (1):

- Capa de gestión de elemento de red (NE).
- Capa de gestión de red.
- Capa de gestión de servicio.
- Capa de gestión de negocio.

Las OSF en estas capas interactúan con las OSF en la misma capa u otras capas dentro de la misma TMN a través de un punto de referencia “q3”, y con aquellas de otra TMN a través de un punto de referencia “x”. Adicionalmente, la gestión de un elemento de red está basada en los datos colectados acerca de los respectivos elementos de la red, los elementos de red en sí mismos pueden ser considerados como la capa más baja en la jerarquía. En contraste con las cuatro capas de OSF, la capa de los elementos de red no involucra OSF, pero está relacionada con la NEF (1).

Los elementos de red son componentes básicos de la red gestionada, instalados como dispositivos físicos, especificados por funciones e interfaces estándares, capaces de distribuir datos en su operación, y proveer medios para ser controlados en una forma específica por el sistema de gestión.

La capa de gestión de elemento de red gestiona cada elemento de red sobre una base individual ó en grupo. La gestión de NE incluye la reunión de datos de cada uno de los elementos de red y el control individual de ellos. En esta capa, las decisiones sobre el cambio de estado de cualquiera de los NE, individualmente, deben basarse en información acerca del mismo elemento, y no puede depender del estado de cualquier otro de los NE ó del estado de la red entera.

La gestión básica de fallas, así como las operaciones de gestión de rendimiento, tales como monitoreo y muestra de condiciones de faltas ó rendimiento de tráfico de cualquiera de los elementos simples de red, así como la toma de acciones elementales para eliminar un error son ejecutados por la capa de gestión de NE.

La capa de gestión de red tiene la responsabilidad por la gestión de la red completa como es soportada por la capa de gestión de elemento. Esta capa transgrede la competencia de la gestión de elemento de red y es responsable por la interconexión y cooperación de todos los elementos de red en el sistema gestionado.

Las tareas de esta capa incluyen gestión de configuración, gestión de eventos, fallas, y rendimiento a nivel de red, así como la gestión de seguridad.

La gestión de servicio se relaciona y es responsable de los aspectos contractuales de los servicios provistos a clientes ó disponibles para potenciales nuevos clientes. La gestión de servicio apunta a establecer relaciones entre los servicios provistos por la red y los requerimientos de los usuarios ó clientes. Los clientes y los contratos de servicio son grabados, los clientes y los parámetros de servicio apropiados son relacionados, se traza la calidad de servicio, las quejas de los clientes son reportadas, y nuevas órdenes son aceptadas y procesadas en esta capa de gestión.

La gestión de negocio es responsable por la empresa total. Tiene que ver con aspectos técnicos y de negocios en función de un complejo orgánico de la actividad de los operadores de red. Las funciones incluidas en esta capa son tarificación y contabilidad (accounting), gestión de mantenimiento, control de costos, control de inventario de repuestos, diseño de nuevos elementos de red y/o nuevos servicios de red, modelado

técnico y optimización, y planeamiento y evaluación de réditos de nuevas inversiones, etc.

La capa de gestión de negocio incluye funcionalidad propietaria. A efectos de prevenir acceso a su funcionalidad, las OSF en la capa de gestión de negocios no tienen generalmente puntos de referencia “x” e interfaces “X”. Esta capa se incluye en la arquitectura TMN para facilitar las especificaciones requeridas por las otras capas de gestión. Sin embargo, aún puede ser necesario que la capa de gestión de negocios interactúe con otros sistemas de gestión ó información. La tarea de estas interacciones debería ser resuelta por soluciones de software.

Las capas lógicas de la jerarquía de gestión no están definidas en total detalle por los estándares existentes. Las tareas y procesos de las diferentes capas y las reglas de intercambio de datos entre ellas no están exactamente determinados al presente. La estructura de capas está esquemáticamente cubierta por las recomendaciones M.30 y M.3010.

En la literatura técnica, la discusión de funcionalidad de gestión de red es a menudo confinada a un simple listado de funciones de gestión sin detalle de sus jerarquías. En la práctica presente, las soluciones estándares están restringidas a realizar las capas de gestión de elemento de red y gestión de red. Frecuentemente usadas, funciones de gestión estándar esencialmente corresponden a las tareas de estas dos capas de gestión. Todavía, relativamente pocas compañías ofrecen soluciones reales para la gestión de servicio. En algún grado, productos de gestión de red apropiados para ejecutar tareas de gestión de negocios están ahora mayormente bajo desarrollo.

Como consecuencia, las herramientas existentes prometen resolver las tareas de más alto nivel de la gestión de red y pueden ser consideradas como productos de software propietario, específico del fabricante.

1.1.1 Proceso actual y propuesta de solución

La gestión de fallas ha sido tradicionalmente definida como la detección, diagnóstico y corrección de una falla o problema. Típicamente, el sistema es monitoreado para permitir una detección oportuna del incidente.

La implicación de este proceso, es que de hecho en la práctica se trata de una progresión de eventos:

- El problema o falla es detectado
- El problema o falla es diagnosticado
- Se llevan a cabo las acciones correctivas que apliquen

Esto implica que un problema o falla necesariamente debe ser diagnosticado antes de que se puedan tomar acciones correctivas.

El diagnóstico de un problema en ocasiones puede ser difícil, ya que se debe tomar en cuenta la complejidad de los sistemas y su constante cambio. Los sistemas son complejos por sí mismos, además que poseen una complejidad mayor con las interacciones hacia componentes externos. Los cambios en los sistemas son provocados por

- Actualizaciones de hardware y software del sistema
- Instalación de nuevos componentes del sistema
- Cambios provocados por el medio ambiente

Un diagnóstico típico involucra

- La correlación de información proveniente de diferentes fuentes (la mayoría de los enfoques de correlación implican empatar la información con patrones o modelos de fallas previamente establecidos).
- El nivel al que debe ser correlacionada la información (profundidad)
- La cantidad información que debe ser correlacionada

La información es recibida en diferentes tiempos y la información proveniente de diferentes fuentes puede estar incompleta o faltar en su totalidad; en un ambiente distribuido, los sistemas no están preparados típicamente para compartir información de fallas o desempeño.

Algunos de los puntos comunes en las redes de administración y supervisión, son el filtrado de alarmas; diagnóstico automatizado y aspectos en tiempo real. El proceso de filtrado de alarmas es una funcionalidad que evita que sean reportados al operador eventos no deseados, con la finalidad de que éste se enfoque en el análisis de la información más importante. El diagnóstico automatizado puede lograrse a través de la detección de eventos-reportes previamente establecidos.

Como ejemplo, algunas plataformas de administración de redes ofrecen herramientas muy poderosas al operador con el propósito de apoyarlo a optimizar y automatizar algunas tareas, tales como los sistemas FMX (© Ericsson), EVA (© Nokia Siemens) y NIX TeMIP (© HP).

Los sistemas expertos antes mencionados son sistemas propietarios que sólo administran eventos y alarmas provenientes de equipos y tecnologías de sus propios fabricantes. Por lo que la integración de nuevas plataformas o tecnologías resulta muy complicado y con precios muy elevados.

Como una alternativa de solución, se propone implementar un Sistema Experto que permita el manejo de eventos y alarmas en un ambiente con tecnologías heterogéneas.

El Sistema deberá aplicar reglas en “tiempo real” para administrar el desempeño y la calidad de los sistemas de telefonía, que automaticen el proceso de administración de

fallas de una red. El módulo de reglas del sistema deberá tener la función de capturar el conocimiento y la experiencia de los administradores de la red y aplicará este conocimiento en la administración de las diferentes redes que así lo requieran.

Cuando un evento ocurre en la red, el sistema ejecutará acciones inteligentes y expertas. Ocultará alarmas con prioridad baja, que permitirá a los operadores de la red enfocarse solamente en aquellos eventos que tengan un impacto en la integridad de la red. El sistema proveerá información detallada de alarmas específicas y recomiendan las acciones correctivas.

Como paso fundamental para dicho sistema es necesario el diseño y desarrollo de una máquina de inferencia capaz de realizar sus funciones de razonamiento que cumpla con las restricciones temporales que implican el manejo de fallas en una red TMN.

Una vez teniendo dicha máquina, será posible utilizar adaptadores de datos para que ésta pueda interactuar con redes de diferentes tecnologías e integrar todo el proceso de correlación en un sistema centralizado.

1.2. Objetivo

Desarrollar una máquina de inferencia en tiempo real para un sistema experto de detección y atención a alertas en una red de telecomunicaciones, así como realizar pruebas de su eficacia y eficiencia.

1.3. Relevancia y contribuciones

Este trabajo es relevante porque se diseñará y desarrollará un modelo de una máquina de inferencia en tiempo real para un sistema experto que permita la administración de alarmas en redes de telecomunicaciones en línea a través del manejo de eventos en una estructura multi-hilos, modular y abierta, y su implementación como una herramienta ad-hoc para la construcción y evaluación de reglas para la gestión de fallas en redes de telecomunicaciones en el sector público o privado. Todo esto en un ambiente con tecnologías heterogéneas, a diferencia de las soluciones empleadas por proveedores de equipos de telecomunicaciones que están enfocadas a una familia de tecnologías únicamente.

En la actualidad existen varias arquitecturas y modelos de sistemas expertos tanto comerciales como en software libre que están orientados a problemas de carácter general; por lo cual la contribución principal de este trabajo radica en el diseño de una máquina de inferencia en paralelo y que trabaje en tiempo real con aplicación para el sector de redes de telecomunicación.

2 Marco teórico y estado del arte

2.1 Sistemas Expertos

Las herramientas de la Inteligencia Artificial pueden dividirse en Inteligencia Computacional, Sistemas Basados en Conocimientos (SBC) y sus híbridos. Los SBC basan su funcionamiento en representaciones simbólicas, es decir reglas explícitas que pueden ser escritas y leídas por humanos. A diferencia de los SBC La Inteligencia Computacional basa su funcionamiento en valores numéricos que cambian de magnitud y se ajustan para acercarse a resultados esperados.

Un sistema basado en conocimientos se diferencia de un sistema convencional en el que el conocimiento referente al dominio de aplicación constituye un rol diferente al funcionamiento del software. En su forma más simple, un SBC se encuentra constituido por 2 módulos: una base de conocimientos y un módulo control (máquina de inferencia).

La base de conocimientos es la entidad que se encarga de almacenar los conocimientos adquiridos. Cabe señalar que el termino conocimiento suele utilizarse indistintamente de otros tales como información, y datos. Es conveniente establecer una jerarquía del conocimiento para especificar el significado de algunos términos.

- Ruido Es cualquier alteración carente de significado definido. (2)
- Datos Son aquellos elementos filtrados del ruido y que son potenciales de ser usados. (2)
- Información Son los datos procesados y utilizados. (2)
- Conocimiento Es la información utilizada por agentes para resolver algún problema. (2)
- Meta conocimiento Es el conocimiento sobre el conocimiento lo cual nos permite manejarlo. (2)

Los conocimientos se van a encontrar organizados y representados de forma que puedan ser interpretados por una computadora. Al proceso de reunir experiencia, organizarla en una estructura computacional y construir bases de conocimiento se le llama Ingeniería del Conocimiento.

Para construir la base de conocimientos el ingeniero de conocimientos puede entrevistarse con la gente poseedora de los mismos para llegar a acuerdos referentes a aspectos relevantes y a la generación de reglas representativas del domino de estos conocimientos. Esto tiene el inconveniente de verse afectado por problemas u errores de comunicación.

Otra opción es que el ingeniero de conocimiento adquiera los conocimientos por su cuenta, eliminando la posibilidad de errores relacionados con la comunicación. Obviamente ésta solución no es práctica en caso de problemas complejos o muy especializados.

Finalmente se podría diseñar un mecanismo de aprendizaje para que el sistema genere sus propias reglas y conocimientos a partir de ejemplos u ejercicios de entrenamiento.

La máquina de inferencia se encarga de llegar a soluciones imitando el proceso de razonamiento humano en un área en particular, valiéndose de los conocimientos representados en la base de conocimientos. Esta actividad se lleva a cabo mediante mecanismos de inferencia, algunos de los cuales son:

- Encadenamiento hacia adelante: Se toma toda la información disponible, a partir de la cual se generan tantos hechos derivados como sea posible. Este proceso se utiliza para problemas de interpretación, donde se llega a una conclusión basada en los hechos que concuerden con las premisas establecidas en la base de conocimientos. Este encadenamiento emplea modus ponens para que a partir de premisas atómicas se obtengan inferencias del tipo situación=> respuesta.

- Encadenamiento hacia atrás: Consiste en partir de una premisa objetivo, como puede ser un enunciado, y las reglas que llevan a dicho objetivo. El mecanismo trabaja hacia atrás relacionando las reglas con la información de la base de conocimientos hasta que el objetivo pueda ser verificado o negado.

Una vez definida la naturaleza de un SBC es posible dar el concepto de Sistema Experto (SE). Un SE es un tipo de SBC diseñado para actuar a semejanza de un experto humano que puede ser consultado en una gran variedad de problemas incluidos en su especialidad (3).

Una definición complementaria encontrada en Marcellin(4), dice que Un Sistema Experto (SE) es un sistema basado en computadora que integra bases de datos, memorias, mecanismos de razonamiento, agentes, algoritmos, heurísticas, para adquirir, representar, almacenar, generar y difundir conocimientos, inicialmente adquiridos a través de varios expertos humanos dentro de un dominio específico llamado “nube”.

Con un Sistema Experto, se pueden dar recomendaciones y/o tomar acciones en las áreas de análisis, diseño, diagnóstico, planeación y control o dar solución a problemas o aplicar técnicas de enseñanza o en general recomendar, actuar y explicar las acciones que hay que tomar en actividades en las cuales normalmente, se requiere del conocimiento o saber de expertos humanos dentro de un dominio específico (4).

Los sistemas expertos se diseñan para funcionar en un dominio de problema, es decir el área específica en la que un especialista puede resolver problemas. Al conocimiento del especialista para resolver problemas determinados se le llama dominio de conocimientos del experto (5).

La programación basada en reglas es una de las técnicas más utilizadas para el desarrollo de sistemas expertos. Un sistema experto basado en reglas consiste de un conjunto de reglas que pueden ser repetidamente aplicadas a una serie de hechos, teniendo en cuenta que:

- Los hechos representan circunstancias que describen una situación en el mundo real.

- Las reglas representan heurísticas que definen un conjunto de acciones a ser ejecutadas en una situación particular.

Podemos tener dos tipos de reglas:

- Las reglas orientadas a datos o reglas de producción son declarativas, se parte de una descripción declarativa de la situación de disparo (una consulta) sin una definición exacta de porque o cuando es detectada esta situación. La forma en que se expresan estas reglas es la siguiente

  SI condición Entonces acción (6)

- Las reglas Orientadas a eventos o Evento-condición-acción, son orientadas a procedimientos porque definen explícitamente sus situaciones de disparo. Específicamente son disparadas por un evento que ocurre dentro o fuera del sistema, luego se revisa para verificar el contexto de disparo y finalmente las acciones son ejecutadas. Este tipo de reglas son expresadas de la siguiente manera:

  En evento Si condición Entonces acción (6)

Los sistemas basados en reglas con encadenamiento hacia adelante funcionan mediante el ciclo Emparejamiento-Selección-Acción (7). Este ciclo consta de las fases de Emparejamiento, Selección y Acción, que se describen a continuación de manera breve.

- Durante la fase de Emparejamiento el sistema verifica cuales condiciones de las reglas corresponden con datos en la memoria de trabajo. Cada regla cuyas variables han sido reemplazadas por valores reales es llamada instancia.

- En la fase de Selección el sistema selecciona una sola instancia del conjunto de instancias emparejadas, basándose en un criterio de resolución de conflictos. Finalmente la instancia es ejecutada en la fase de Acción. (6)

- En Acción la regla es ejecutada, esto puede involucrar el cambio en el contenido de la memoria de trabajo. Al concluir esta fase se ejecuta la fase Emparejamiento y comienza el nuevo ciclo.

Uno de los principales cuellos de botella dentro de los sistemas basados en reglas consiste en la fase de Emparejamiento (6). El método ingenuo consiste en evaluar regla por regla contra cada hecho en la "memoria de trabajo", esto implica problemas de desempeño, ya que se evalúan hechos que pueden no haber sido modificados. Para mejorar el tiempo de respuesta han surgido algoritmos como RETE (8), LEAPS (7)y TREAT (9).

De estos algoritmos, RETE y TREAT se basan en la compilación de las reglas en una grafo llamado "red discriminante", que acepta como entrada los cambios ocurridos en la memoria de trabajo y entrega a la salida la instancia que debe ser añadida o removida del "conjunto de conflicto", el cual contiene las reglas preseleccionadas para su ejecución. (6)LEAPS por otra parte se basa en estructuras de datos complejas y algoritmos de búsqueda en lugar de una red discriminante (7).

2.2 Sistemas en tiempo real

En los sistemas en tiempo real, el buen funcionamiento no sólo depende de obtener un algoritmo correcto, sino que además el programa debe de cumplir con un plazo máximo de terminación para entregar un resultado. Se dice que un sistema tiene requerimientos en tiempo real estricto (hard) si al no cumplirse con el plazo la utilidad del sistema es nula o altamente degradada. Una tarea en tiempo real no estricto (soft) ocurre cuando la utilidad del sistema disminuye al no cumplirse con un plazo máximo de ejecución (10).

El espacio de problemas en tiempo real tiene tres dimensiones a considerar, la precisión con la que se mide el tiempo, la importancia de la consistencia y la curva de utilidad del sistema alrededor del plazo (10).

La precisión con la que se mide el tiempo es esencial para caracterizar el sistema. Así por ejemplo un controlador de hardware requiere respuestas en unos cuantos microsegundos, y un sistema en tiempo real de propósito general puede operar en el rango de milisegundos. Los sistemas en tiempo real distribuidos deben lidiar con la

velocidad de comunicación de la red, por lo que generalmente trabajan en centésimas de segundo. Finalmente los programas que interactúan con personas especifican el tiempo en décimas de segundo (10).

En un entorno en tiempo real se requiere trabajar en un ambiente consistente, es decir tratar de reducir la diferencia entre el mejor tiempo de ejecución y el peor. Esta consistencia involucra no obtener siempre el mejor rendimiento posible, ya que no es deseable utilizar heurísticos y algoritmos que podrían acelerar el mejor caso posible o incluso el caso promedio, pero disparar el tiempo de ejecución del peor caso (10).

2.3 Sistemas expertos en tiempo real

En su tesis Emilio(11) propone una definición de un sistema de inteligencia artificial en un entorno en tiempo real como "un proceso de búsqueda con restricciones temporales en el espacio de estados del problema". Existen tres formas principales para combinar técnicas de inteligencia artificial y tiempo real en un sistema único (12): Embebiendo la inteligencia artificial en un sistema en tiempo real. Embebiendo reacciones en tiempo real en un sistema de inteligencia artificial. Acoplando la inteligencia artificial y el sistema en tiempo real como sistemas paralelos

Siendo el primer enfoque el más simple, consistiendo en integrar métodos de inteligencia artificial en un sistema en tiempo real, forzándolos a que cumplan con los plazos de tiempo predeterminados. El problema de este enfoque es que las técnicas de inteligencia artificial, generalmente no son muy apropiadas para los mecanismos de planificación de tiempo real, que requieren de conocer el peor tiempo de ejecución para cada tarea. Mientras que las tareas en tiempo real pueden tener tiempos de ejecución cortos, existen muchas variaciones debidas a búsquedas y dependencia de datos. Existen dos formas de tratar aliviar este problema, reducir las variaciones en el tiempo de ejecución, o bien los algoritmos son llamados de forma incremental e interrumpible (12).

Existen 2 principales cuellos de botella en la ejecución de reglas de producción, el principal que es la etapa de Emparejamiento dentro del ciclo de Emparejamiento-Selección-Acción y el otro consiste en la ejecución de una regla por ciclo de producción (6).

Para evitar estos cuellos de botella surgieron técnicas como RETE que evitan la evaluación de reglas que no pueden ser aplicadas, o bien crear lazos internos entre las reglas, mediante usos de punteros y variables (13).

En este trabajo, se opta por explorar el camino de paralelizar el proceso de emparejamiento y ejecución de reglas.

2.3.1 Emparejamiento de reglas en paralelo

Cómo se mencionó en la sección 2.2 varias técnicas para el procesamiento y mejora de desempeño han surgido, entre ellas la red discriminante, sin embargo una mejora adicional sólo puede ser lograda paralelizando el proceso en un ambiente multiprocesador (6). Existen varios esquemas para paralelizar el proceso de emparejamiento de reglas:

- Mapear la red discriminante en una arquitectura multiprocesador asignando cada nodo de la red a un procesador.

- Colocar únicamente los nodos de dos entradas (nodos que realizan la operación de unión entre condiciones individuales) en diferentes procesadores.

- Distribuir las reglas de producción entre un número de procesadores, los cambios en la memoria de trabajo son distribuidos a todos los procesadores y el emparejamiento se realiza en paralelo en los procesadores (6).

2.3.2 Ejecución de reglas en paralelo

En el caso más general cada instancia de una regla en el "conjunto conflicto" (lista de reglas seleccionadas para su ejecución) puede ser ejecutada en paralelo (6). Sin embargo existe una condición a ser tomada en cuenta, la seriabilidad.

La seriabilidad restringe el paralelismo para garantizar que los resultados de la ejecución de reglas son equivalentes a al menos una ejecución secuencial de reglas. Las reglas no son secuenciales cuando interfieren entre sí (6).

Existen dos casos de interferencia de reglas:

- Lectura-escritura: Cuando la acción de una regla modifica un elemento de la memoria de trabajo utilizado como condición de otra regla.

- Escritura-escritura: Cuando dos reglas modifican con sus acciones el mismo elemento de la memoria de trabajo de manera contradictoria (6).

Un acercamiento para aliviar este problema consiste en la elaboración de un grafo de dependencias en el momento de compilación de la base de reglas. Este grafo permite identificar las interferencias de las reglas y prohibir la ejecución concurrente de las instancias que interfieren entre sí (14).

Otro enfoque consiste en emplear un mecanismo de "bloqueo" (15). Cada transacción obtiene un "seguro" de lectura para los elementos de la memoria de trabajo de los cuales depende, igualmente al momento de escribir sobre un elemento de la memoria de trabajo (en la fase Acción), la transacción adquiere un "seguro" de escritura sobre ese elemento.

Si alguna otra transacción para una regla requiere utilizar dicho elemento debe esperar a que el seguro sea retirado por la transacción que primero adquirió el "seguro" (6). Este esquema tiene la ventaja de que siempre funciona de la misma manera, mientras que el análisis de las reglas no siempre proporciona resultados precisos (6).

2.4 Correlación de eventos y manejo de fallas

Como se mencionó en la sección 2.1, una red TMN es un sistema distribuido altamente complejo, una gestión de fallas eficiente dentro de la capa de gestión de red del OSF requiere de un adecuado manejo de los eventos (alarmas provenientes de los equipos). Un fenómeno común es una "lluvia" de eventos, es decir, los efectos de una falla son detectados por varios equipos los cuales son reportados simultáneamente al operador, comúnmente impidiendo la debida atención a eventos importantes debido a la gran cantidad de eventos presentados al operador.

Estos problemas son atendidos por sistemas conocidos comúnmente como herramientas de correlación de eventos, la forma más sencilla de este tipo de sistemas consiste en la aplicación de mecanismos clásicos de filtrado, pero estas técnicas no son lo suficientemente sofisticadas para correlacionar secuencias complejas de eventos.

En un sistema de correlación de eventos se reciben un conjunto de eventos relacionados, se condensan y se notifica al operador. El componente principal en este sistema consiste en un repositorio del la herramienta de correlación obtiene información de cómo están relacionados los eventos (16).

Una herramienta de correlación de eventos se ocupa de tres problemas:

1. Reducir el número de eventos para impedir que se presenten "tormentas de eventos" al operador del sistema.

2. Los eventos generados por la herramienta de correlación deben indicar la causa del problema en lugar de los síntomas.

3. Los eventos generados deben de ser mandados a la instancia correcta para su atención (tomar una acción, notificación al operador indicado o reingreso al sistema dependiendo del caso).

Un grupo de enfoques para el desarrollo de estos sistemas consiste en la creación de un conjunto de casos. Los eventos llegan al sistema cómo un conjunto de síntomas, el sistema busca similitudes en la base de casos y presenta las soluciones almacenadas en los casos cuyos síntomas concuerdan con los recibidos. El principal problema de estos

sistemas consiste en tener una buena medida de similitud y en tener una buena base de casos (16).

Otro enfoque consiste en permitir a los expertos plasmar su experiencia en una base de conocimientos, a través de reglas de la forma "SI eventoA Y eventoB EMITE eventoC" (16). Este enfoque es el elegido para el diseño y desarrollo de un sistema inteligente para el manejo de fallas en el presente trabajo ya que permite aprovechar la experiencia de los operadores, quienes pueden decidir cuáles son las alarmas pertinentes de análisis y correlación dotando al sistema de gran flexibilidad y adaptación a diversos ambientes y tecnologías. Claro un inconveniente consiste en que cada instancia del sistema contendrá un conjunto diferente de reglas si no existe consenso entre los expertos responsables del mantenimiento de cada sistema.

2.5 Estado del Arte

2.5.1 Máquinas de inferencia y sistemas expertos

ILOG una compañía internacional de software perteneciente a IBM (17). Entre sus productos está ILOG Rules, un ambiente para el desarrollo de sistemas expertos. Provee una máquina de inferencia en C++ y Java utilizando una variante optimizada del algoritmo RETE (18).

La compañía Gensym desarrolla la plataforma G2. La cual ha servido en la automatización de procesos en diferentes campos. Incluye tecnologías de razonamiento en tiempo real como reglas, flujos de trabajo, procedimientos, simulación y graficación. Es orientado a objetos con fuertes características en tiempo real (19).

CLIPS es un ambiente de desarrollo para sistemas expertos. Fue creado en 1984 en el Lyndon B. Johnson Space Center. Soporta programación lógica, programación imperativa y programación orientada a objetos. Es ampliamente usado debido a su eficiencia y a que es gratuito (20). Está basado en el algoritmo RETE.

Production System Technologies ofrece CLIPS/R2, una implementación de CLIPS pero cuyo algoritmo de razonamiento está basado en RETE2, algoritmo propuesto por Forgy y cuyo exacto funcionamiento no ha sido desvelado (21).

Jess es una máquina de inferencia y lenguaje de scripting escrito en lenguaje Java. Utiliza una versión mejorada del algoritmo RETE. Tiene funciones únicas como encadenamiento atrás y búsquedas en la memoria de trabajo (22).

Drools Expert de JBoss es una máquina de inferencia en Java, también utiliza el algoritmo RETE adaptado con técnicas de indexación, hashing y elementos del algoritmo LEAPS (23) (24).

2.5.2 Manejo de fallas en redes de telecomunicaciones

"HP OpenView TeMIP" es una familia de productos de software para el manejo de redes de telecomunicaciones "HP OpenView TeMIP Expert" es un sistema experto dentro de esta familia. Permite realizar acciones de alto nivel en una plataforma TeMIP como:

- Filtrado de alarmas.
- Generación de alarmas.
- Enriquecimiento de las alarmas.
- Análisis de causa.
- Está basado en el ambiente de desarrollo para sistemas expertos "ILOG Rules" (18).

"Fault Manager Expert FMX" de Ericsson que automatiza el proceso de administración de fallas. Está basado en la plataforma "G2" de Gensym. Toma conocimientos expertos, resalta alarmas importantes, y oculta las que no lo son. Esto permite a los operadores enfocarse sólo en los eventos que impactan a la red directamente. Atiende 500000 alarmas por día para 50 tipos de equipo diferentes (25).

Teniendo ya una clara perspectiva de las necesidades que cubre un sistema experto en tiempo real para manejo de fallas en un sistema experto de comunicaciones se procede a diseñar la máquina de inferencia para dicho sistema.

3 Diseño de una máquina de inferencia para un sistema experto de telecomunicaciones

Antes de entrar al diseño de la máquina de inferencia es necesario hacer un diseño, aunque sea elemental, del sistema experto para el manejo de alertas en una red de telecomunicaciones.

3.1. Diseño de una arquitectura de un sistema experto para el manejo de alarmas en una red de telecomunicaciones

Se parte de la base de que el sistema experto es sólo un apoyo al sistema de administración de fallas AF, el AF se encargará de recibir los mensajes de error provenientes de la TMN, los procesará y traducirá a un formato estándar. Además se encargará de almacenar las alarmas en una base de datos con un registro histórico de los eventos acontecidos en la red.

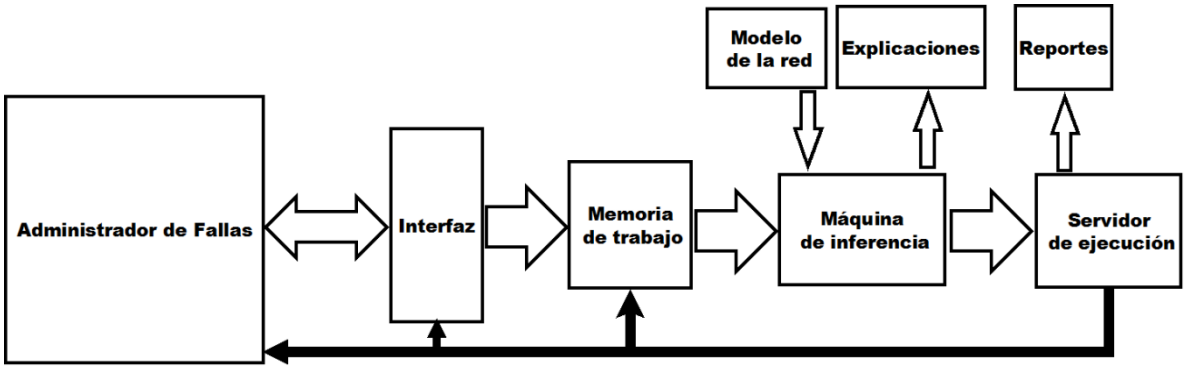

Figura 3.1: Arquitectura de un sistema experto para manejo de alarmas en una red TMN.

Una vez que se ha estudiado este contexto podemos plantear un diseño general del funcionamiento e interacciones del sistema experto, como se aprecia en la figura 3.1.En esta figura se destaca los siguientes módulos:

Administrador de fallas: Sistema que se encarga de recibir las alarmas de diferentes elementos de la red, pre-procesarlas, almacenarlas en una base de datos y mandarlas a los módulos de interfaz con la máquina de inferencia.

Interfaz: Elemento que se encarga de recibir los mensajes de alarma del administrador de fallas y traducirlos a un formato entendible por la máquina de inferencia. Este módulo existe en caso de que se interactué con 2 o más sistemas de manejo de fallas heterogéneos. Así no será necesario cambiar la arquitectura de la máquina de inferencia, simplemente se añadirán funciones al módulo de interfaz.

Memoria de trabajo: Los eventos de alarma recibidos por el módulo de interfaz son recibidos por el módulo de memoria de trabajo, este consta de una lista de hechos ordenados por tiempo de llegada y que residirán en dicha memoria por un intervalo de tiempo definido y cuyo estado sea activo, esto se explicará más adelante.

Modelo de la Red: es una base de datos que contiene la información de la topología de la red, conexiones, grupos de trabajo y demás relaciones entre los elementos de la red, esta información es semi-estática (es necesario actualizarla cuando se realiza un cambio en la red). Sirve de apoyo en el proceso de inferencia.

Módulo de explicaciones: Este módulo se encargará de recibir y almacenar la "traza" de ejecución de las reglas con el fin de poder justificar las conclusiones tomadas durante el proceso de inferencia. Dicha traza puede ser presentada al usuario para evaluación y obtención de estadísticas del sistema.

Módulo de reportes: Se encarga de recibir las conclusiones obtenidas de la máquina de inferencia y registrar las acciones tomadas para proceder a presentarlas al operador.

Máquina de inferencia: procederá a realizar el proceso de inferencia a partir de los eventos detectados, los eventos en la memoria de trabajo y el modelo de la red.

Servidor de acciones: Recibe las conclusiones obtenidas de la máquina de inferencia y procede a tomar una acción, como puede ser generar, actualizar o eliminar una alarma de la memoria de trabajo o el administrador de fallas, para esto es necesario que se comunique con el módulo de Interfaz y la Memoria de trabajo.

3.2. Diseño de la máquina de inferencia.

Primero se presentan los requisitos identificados para una máquina de inferencia en tiempo real para un sistema de manejo de fallas en una red de telecomunicaciones:

- Atender 120 mensajes de alarma por minuto.

- Ser capaz de trabajar con una base de conocimiento de 50 reglas en paralelo sin que exista un decaimiento en el rendimiento del sistema.

- La entrada al sistema está constituida por mensajes de alarma, y otro tipo de eventos (eliminación o actualización de una alarma).

- La salida consta de creación, eliminación o actualización de mensajes de alarma.

- Debe llevar un registro de acciones tomadas, así como de la secuencia de activación de las reglas. Estos registros se almacenarán en una tabla contenida en una base de datos.

- Las reglas incluyen un conocimiento de la topología de la red, por lo que debe existir una representación de dicha topología con la cual se pueda comunicar la máquina de inferencia para realizar consultas.

A continuación se presenta el diseño de los componentes del sistema necesarios para atender cada una de las necesidades y requisitos identificados. Se inicia por la representación de la topología de la red.

3.2.1. Diseño de la representación de la topología.

La ITU (International Telecommunication Union) presenta en su estándar M.3100 el modelo de información genérico, el cual, identifica clases de objetos de la TMN que son comunes a redes de telecomunicaciones gestionadas; o que son superclase de otras clases; o que son elementos de gestión requeridos para la red de telecomunicaciones. Sirve para identificar los recursos genéricos existentes así como sus tipos de atributos, eventos acciones y comportamientos asociados. Este modelo está especificado a través de técnicas de modelado UML. Es un modelo orientado a objetos, donde objetos con atributos y comportamientos similares se agrupan en clases. Las clases de objeto y tipos de atributo son definidos sólo a fin de comunicar mensajes de gestión de red entre sistemas, y no tienen por qué estar relacionadas con la estructuración de datos dentro de dichos sistemas.

Para el siguiente trabajo se toma en cuenta que existe una representación del modelo de información como parte del sistema de gestión de fallas. Esta representación puede estar estructurada en una base de datos relacional u orientada a objetos.

Una vez con el modelo de información se puede representar la topología de la red, idealmente, cada vez que se realiza un cambio en el modelo de información, se realizará un cambio en la representación de la topología. La elaboración del subsistema encargado de generar dicha topología está fuera del alcance de este proyecto.

La topología consistirá en un modelo orientado a objetos que sólo se interesará en relaciones de conexión, contención y herencia entre los objetos. Se optó por una representación en formato XML de la topología, para ello se creó el un esquema XML para definir la estructura de esta representación XML. El esquema nos permitirá validar que una representación de la topología esté de acuerdo a una estructura predeterminada. Se utilizó como inspiración el formato presentado por la máquina de simulación de tráfico “TOTEM project (TOolbox for Traffic Engineering Methods)” (26), con modificaciones para adaptarla a una arquitectura orientada a objetos. A continuación se presenta dicho esquema:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://tempuri.org/XMLSchema2.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Topology">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nodes">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="object">
                <xs:complexType>
                  <xs:sequence minOccurs="0">
                    <xs:element name="objectRef">
                      <xs:complexType>
                        <xs:simpleContent>
                          <xs:extension base="xs:unsignedByte">
                            <xs:attribute name="name" type="xs:string" use="required" />
                          </xs:extension>
                        </xs:simpleContent>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="id" type="xs:unsignedByte" use="required" />
                  <xs:attribute name="classname" type="xs:string" use="required" />
                  <xs:attribute name="type" type="xs:string" use="optional" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="links">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="link">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="from">
                      <xs:complexType>
                        <xs:attribute name="id" type="xs:unsignedByte" use="required" />
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="to">
                      <xs:complexType>
                        <xs:attribute name="id" type="xs:unsignedByte" use="required" />
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="id" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

El esquema representa un nodo raíz llamado “Topology” este nodo contiene dos ramas principales “nodes” que contendrá una representación de los objetos de la topología. Aquí se encuentra información relativa a la herencia y la contención de objetos. Cada objeto en “nodes” esta identificado por una etiqueta <object> con atributos “id” que es el nombre del objeto, “classname” la clase a la que pertenece el objeto, y “type” que representa el tipo de objeto, este atributo es el empleado para representar la

herencia entre clases, así si un objeto es de un tipo, indica que la clase a la que pertenece hereda de otra clase especificada en dicho atributo.

En “links” encontramos las conexiones entre los objetos de la red. Cada conexión corresponde a una etiqueta <link>, cada conexión consta de dos nodos, uno origen denotado por <from> y uno destino <to>. Como atributo de dichas etiquetas estará el id de los objetos conectados. A continuación se presenta un ejemplo de la representación xml de la topología.

```
<?xml version="1.0" encoding="utf-8" ?>
<Topology xmlns="http://tempuri.org/TipologyExample.xsd"
id="topologia1">
  <nodes>
    <object id="01" classname="clase1">
      <objectRef name="ObjectRefVar">12</objectRef>
    </object>
    <object id="02" classname="clase2" type="clase1"></object>
  </nodes>
  <links>
    <link id="link1">
      <from id="01"/>
      <to id="02"/>
    </link>
  </links>

</Topology>
```

Aquí primero se define un objeto con id “01” que pertenece a la clase “clase1”. Dicho objeto contiene una referencia a otro objeto, en este caso la referencia a un objeto cuya id es “12” esto implica que el objeto “01” contiene al objeto “12”. Luego definimos al objeto “02” que pertenece a la clase “clase2” y que es de tipo “clase1”, aquí se expresa una herencia, el objeto “02” hereda de la clase a la cual pertenece el objeto “01” . Luego definimos una conexión con id “link1” que va del objeto “01” al objeto “02”.

Con la forma de representar la topología ya definida, es necesario establecer un formato de datos para la máquina de inferencia.

3.2.2. Formato de datos

Los datos de las alarmas contenidos en los registros de alarma y son el componente fundamental para el desarrollo del proceso de inferencia, por ello es necesario elegir y seleccionar un formato de datos flexible y que comprenda la información necesaria para la inferencia. Los registros de alarma generalmente se encuentran representados por una “trama” con varios campos, cada campo sirve para caracterizar un registro y cada registro representa un evento. La figura 3.2 representa un registro de alarma modelo en un sistema de gestión de fallas como se aprecia en (27).

```
%a
-RecordId=2545683
-AlarmState=1
-SystemType=AXE
-EventTime=20060927151200
-ETtext=Quality of service
-EventType=2
-ObjectOfReference=SubNetwork=ONRM_ROOT_MO,SubNetwork=FT_APG,ManagedElement=SEAPG001
-ObjectClassOfReference=21
-BackupObjectInstance=
-TrendIndication=1
-PerceivedSeverity=2
-PCtext=Threshold crossed
-ProbableCause=121
-SPtext=BLOCKING SUPERVISION
-SpecificProblem=10030000
-CorrelatedRecord=2400110
-AlarmClass=3
-AlarmCategory=0
-AlarmId=2317521
-AlarmNumber=1
-AttendanceIndication=0
-RecordType=4
-ObjectType=20
-Acknowledger=
-PRAtext=< Unknown >
-ProblemText=*** ALARM 001 REPEATED O1/APZ "SEAPG001"U 060927 1512
:BLOCKING SUPERVISION
:R        LVB   NDV   BLO
:ALARMA1     8    16    16
:
:END
:
```

Figura 3.2: Registro de alarma típico en el sistema de gestión de fallas de la plataforma Ericsson

Después de analizar algunos formatos de registro de alarma se identificaron los campos necesarios para el proceso de inferencia, pero se realizará un diseño que permita añadir más campos de ser necesario sin tener que reestructurar todo el sistema. Los campos seleccionados se presentan en la tabla 3.1.

Campo	Tipo	Rango	Significado
TipoDeSistema	Int	1	Tipo de sistema al cual pertenece el objeto que emitió la alarma. En este caso todas las alarmas son para el sistema de tipo 1 (caso de prueba).
TipoDeRegistro	Int	1-18	Tipo de registro de alarma 1 = Alarma 2 = Mensaje de error 4 = Alarma repetida 5 = Alarma de sincronización 7 = Sincronización iniciada 8 = Sincronización terminada. 9 = Sincronización abortada. 10 = Sincronización ignorada 18 = Mensaje de error repetido
IdRegistro	Int		Clave identificación del registro de alarma.
Objeto	String		Objeto que emitió la alarma
Severidad	Int	0-5	Severidad de la alarma, a mayor severidad menor es el desempeño del objeto emisor: 0=indeterminada 1=atendida 2=advertencia 3=menor 4=mayor 5=crítica Cuando el objeto indica que la alarma a cesado o bien el sistema asi lo determine, la alarma se da como “atendida”.
Hora	DateTime		Fecha y hora en la que se emitió la alarma
Causa	Int		Causa probable de la alarma, este número dependerá del objeto, por lo que es un rango variable.
ProblemaEspecifico	Int		Problema específco que pudo provocar la alarma, este número dependerá del objeto, por lo que es un rango variable.
Texto	String		Texto adicional para la alarma
NumeroDeAlarma	Int		Numero de identifcación de la alarma, este número identifica a la alarma en el sistema gestor de fallas.
CategoriaDeAlarma	Int		Categoría de la alarma
Status	Int	1-4	Estado actual de la alarma: 1.- Activa, no reconocida 2.- Activa, reconocida 3.- Atendida, no reconocida 4.- Atendida, reconocida
SolucionPropuesta	String		Propuesta de solución al problema

Tabla 3.1: Campos del formato de datos para el registro de alarmas en la máquina de inferencia.

El paso siguiente es diseñar los módulos necesarios para la operación de la máquina de inferencia.

3.2.3 Interfaz entre la máquina de inferencia y el gestor de fallas

Para elaborar la interfaz entre el administrador de fallas y la máquina de inferencia se tienen en cuenta las siguientes características:

- Es un sistema distribuido, el gestor de fallas puede residir en un nodo diferente al sistema experto.

- El gestor de fallas envía mensajes, en forma de registros de alarmas. Estos registros consisten de una cadena de caracteres que contiene los campos especificados en la sección 3.2.2 además de campos adicionales propios a dicho gestor.

- No se requiere una sincronización entre la máquina de inferencia y el gestor de fallas, es decir, el gestor de fallas envía mensajes sin esperar una confirmación de recepción por parte de la máquina de inferencia.

Debido a estas características se decidió adoptar un esquema "productor-consumidor" para la recepción de los registros de alarmas. El productor en este caso es el sistema de administración de fallas, el cual se encuentra generando registros de alarmas que manda al sistema experto. La interfaz recibirá estos registros de alarma, los almacenará en una estructura de datos, dicha estructura será una cola con prioridades, es decir los registros son ordenados dentro de la cola dependiendo al valor de prioridad determinado por el sistema de administración de fallas.

Se eligió una cola con prioridades porque permite mantener la relación temporal entre la detección de una alarma y las subsiguientes, pero a su vez nos da la posibilidad de atender primero las fallas que pueden comprometer en mayor medida la integridad de la red de telecomunicaciones.

Además de la estructura de datos, se identifican los siguientes componentes de la interfaz:

- Un servidor encargado de recibir las comunicaciones del gestor de fallas y almacenarlos en la cola de mensajes para que sean procesados por la máquina de inferencia.

- Un adaptador de datos, para traducir del formato de registro de alarma nativo al gestor de fallas al formato utilizado por la máquina de inferencia y presentado en la tabla 3.1.

- Un programa que lea un mensaje de la cola y lo traduzca a un formato único a través del adaptador de datos para que posteriormente sea utilizado por la máquina de inferencia.

3.2.3.1. Diseño de la Interfaz entre la máquina de inferencia el gestor de fallas.

La interfaz constará de un servidor que recibirá los mensajes provenientes del administrador de fallas, y los mandará a una cola de mensajes. Una vez en la cola los mensajes serán interpretados y mandados a los componentes del bloque de máquina de inferencia que se presentará más adelante. El diagrama de componentes se muestra en la figura 3.3.

El gestor de fallas envía un "Registros de alarma", estos registros llamados RA tienen forma de una cadena de caracteres, los mensajes son enviados a la cola a través de un servidor.

El servidor de cola de mensajes recibe un mensaje y lo almacena en una cola de mensajes, esta cola es priorizada (los mensajes de mayor prioridad se colocan al frente de la cola, en caso de dos mensajes de la misma prioridad se colocan de acuerdo al orden de llegada). A través de un método es posible leer el primer mensaje de la cola, o bien recibir una notificación en cuanto un elemento es añadido a la cola o bien la cola tenga elementos, en cuyo caso se procede a leer el primer elemento de la cola. Este enfoque basado en eventos ofrece la ventaja de evitar que un proceso esté continuamente leyendo la cola en busca de nuevos mensajes.

El control de mensajes es un servicio que lee el mensaje proveniente de la cola, lo traduce a través de un objeto llamado "Parser". Con el fin de dotar a la máquina de inferencia de flexibilidad en formatos, el "Parser" no es fijo, es un objeto que puede ser intercambiado por otro dependiendo del fabricante y formato de datos. Una vez traducido el mensaje se notifica a la máquina de inferencia que se recibió dicho mensaje y es enviado a la misma.

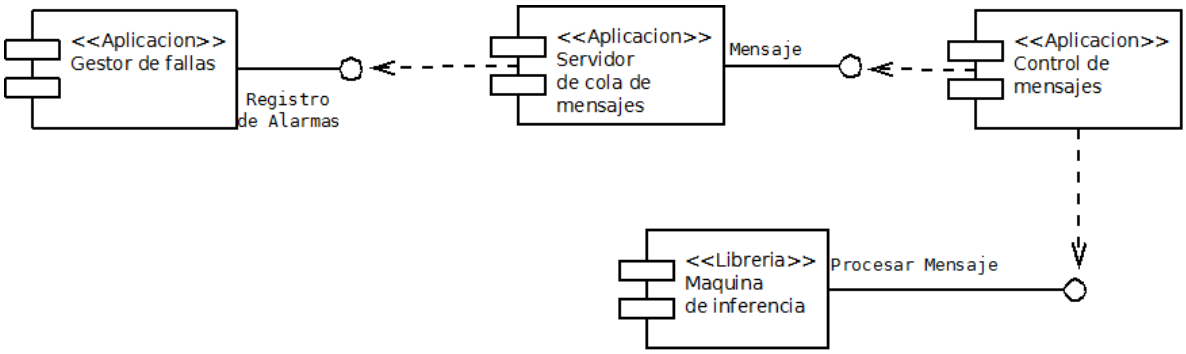

Figura 3.3: Diagrama de componentes de la interfaz entre el gestor de fallas y la máquina de inferencia.

3.2.4. Máquina de inferencia.

Teniendo ya diseñada la interfaz entre la máquina de inferencia y el gestor de fallas se procede a diseñar el mecanismo de inferencia a utilizar. Se eligió un enfoque basado en eventos activadores. Este enfoque como se mencionó en la sección 2.2 consiste en añadir una clausula “Cuando evento” a las reglas de forma “Si entonces”. Así una regla se encontrará de la forma:

“Cuando EVENTO Si CONDICIÓN Entonces ACCIÓN”

Esto debido a que el sistema recibe continuamente mensajes de alerta de manera irregular, un esquema declarativo forzaría a ejecutar una evaluación de reglas en intervalos regulares de tiempo. Se considera que un evento de alarma es el principal determinante del comportamiento del sistema por lo que este esquema basado en eventos es apropiado.

Una vez detectado un evento relevante se procede a revisar el contexto del evento en busca de eventos relacionados en la memoria de trabajo y su posible efecto en otros objetos relacionados al origen del evento, para poder correlacionarlos y concluir las posibles causas y generar nuevos eventos que a su vez pueden activar otras reglas.

Como se explicará posteriormente, este enfoque permite ejecutar el proceso de encadenamiento hacia adelante, además en la evaluación del contexto de la regla se emplea un mecanismo búsqueda en la memoria de trabajo, la topología o la bitácora.

Por lo tanto el funcionamiento de la máquina de inferencia se puede dividir en 2 etapas:

- Una etapa de emparejamiento sobre el evento recibido, se contrasta el evento recibido con los eventos activadores de una regla. Por evento activador se denota a la parte “Cuando Evento” de la regla.

- Una etapa de evaluación de condiciones, en la cual se realizan búsquedas sobre las fuentes de información de la máquina de inferencia para evaluar el cumplimiento de la parte “Si Condición” de la regla.

Se identificaron los siguientes eventos activadores:

- Alarma proveniente del Gestor de fallas: En este caso se revisa que la alarma proveniente del gestor de fallas cumpla con una condición dada.

- Reconocimiento de alarma: Evento que ocurre cuando una alarma se da por reconocida (el operador marca el mensaje de alarma como leído).

- Atención de alarma: Cuando una alarma es eliminada de la memoria de trabajo por una regla.

Una vez identificados los eventos activadores se procede a determinar los objetos sobre los cuales se puede realizar el proceso de inferencia, es decir los bloques válidos para la parte de CONDICIÓN de la regla:

- Evento: el evento activador tiene las características especificadas, como pueden ser atributos, estado de reconocimiento, estado de activación etc. Esto ya es evaluado en la etapa de emparejamiento de la regla, por lo que su inclusión es innecesaria.

- Existe evento: esta condición ocurre cuando se busca un evento particular antes o después de que se disparó una regla en un umbral de tiempo determinado. Sirve para correlacionar eventos.

- Relación entre objetos: se revisa la relación entre dos o más objetos contenidos en la topología, esta relación puede ser de conexión, contención y herencia.

Finalmente se identifican las acciones a tomar:

- Envío de un evento o mensaje: Se genera una alarma en el formato estándar para los registros de alarmas y se envía al servidor de acciones, o bien se envía un mensaje al servidor de acciones.

- Reconocimiento de eventos: Se marca como reconocido un evento de la memoria de trabajo.

- Atención de eventos: Se borra un evento de la memoria de trabajo.

3.2.4.1. Elección de arquitectura

Con el fin de mejorar el rendimiento del proceso de inferencia es necesario elegir una arquitectura, se analizaron las siguientes alternativas:

Emparejamiento de reglas en paralelo con un mecanismo de ejecución síncrono y memoria compartida: Las reglas se dividen en conjuntos disjuntos de reglas, dichos conjuntos son distribuidos a las unidades de procesamiento. Los conjuntos evaluaran las reglas componentes de acuerdo a un algoritmo de emparejamiento de reglas, una vez que se ha emparejado una regla, se manda la conclusión de dicha regla a un mecanismo centralizado, dicho mecanismo esperará a que todas las reglas en ambos conjuntos hayan sido evaluadas y procederá a realizar la tarea de resolución de conflictos para mandar las acciones a realizar al servidor de acciones. Ver figura 3.4

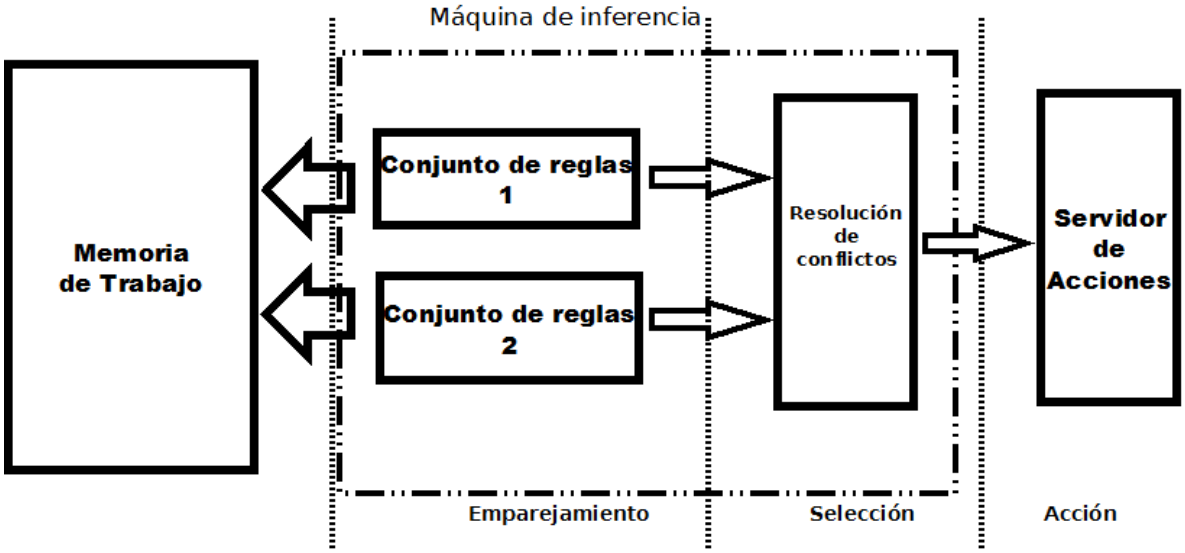

Figura 3.4: Emparejamiento de reglas en paralelo con un mecanismo de ejecución síncrono y memoria compartida

Emparejamiento de reglas en paralelo con un mecanismo de ejecución asíncrono y memoria compartida: Las reglas se dividen en conjuntos disjuntos de reglas, dichos conjuntos son distribuidos a las unidades de procesamiento. A diferencia del mecanismo síncrono, existirá un mecanismo de resolución de conflictos implícito en las reglas, mediante la implementación de seguros y prioridades de reglas, así cada regla se ejecutará en el momento en que acaba de ser procesada, mandando un mensaje al servidor de acciones. Ver figura 3.5 para la estructura general y la figura 3.6 ejemplifica el modelo de ejecución asíncrona de reglas.

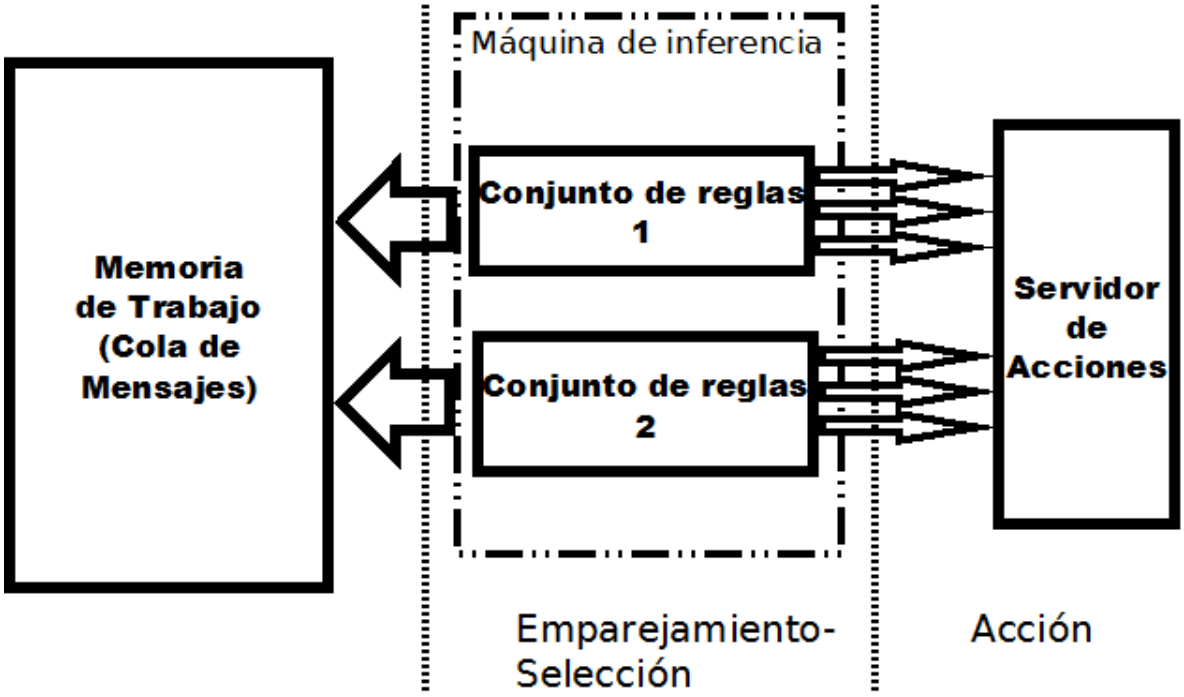

Figura 3.5: Emparejamiento de reglas en paralelo con un mecanismo de ejecución asíncrono y memoria compartida

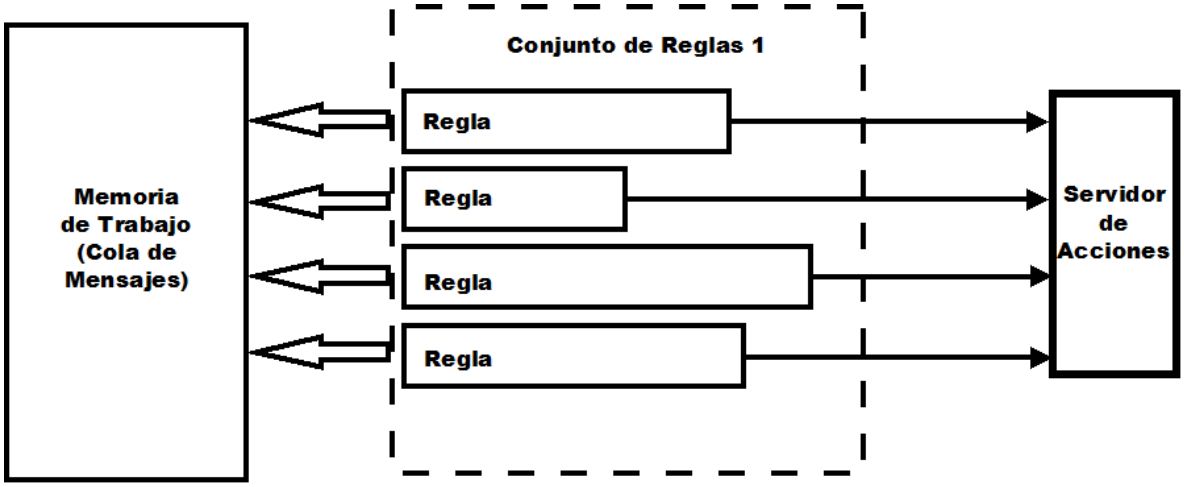

Figura 3.6: Detalle de ejecución asíncrona de reglas

Emparejamiento de reglas en paralelo con un mecanismo de ejecución síncrono y memoria distribuida: Una copia de todas las reglas es mandada a cada procesador para su ejecución. Los mensajes provenientes del AF son mandados a cada procesador, cada procesador se encarga de evaluar las reglas, una vez que ha concluido de evaluar todas las reglas manda un mensaje al mecanismo centralizado junto con las reglas seleccionadas. El mecanismo centralizado recibe mensajes provenientes de cada procesador (reglas seleccionadas) y procede a la resolución de conflictos para proceder a mandar instrucciones al servidor de acciones. Ver figura 3.7

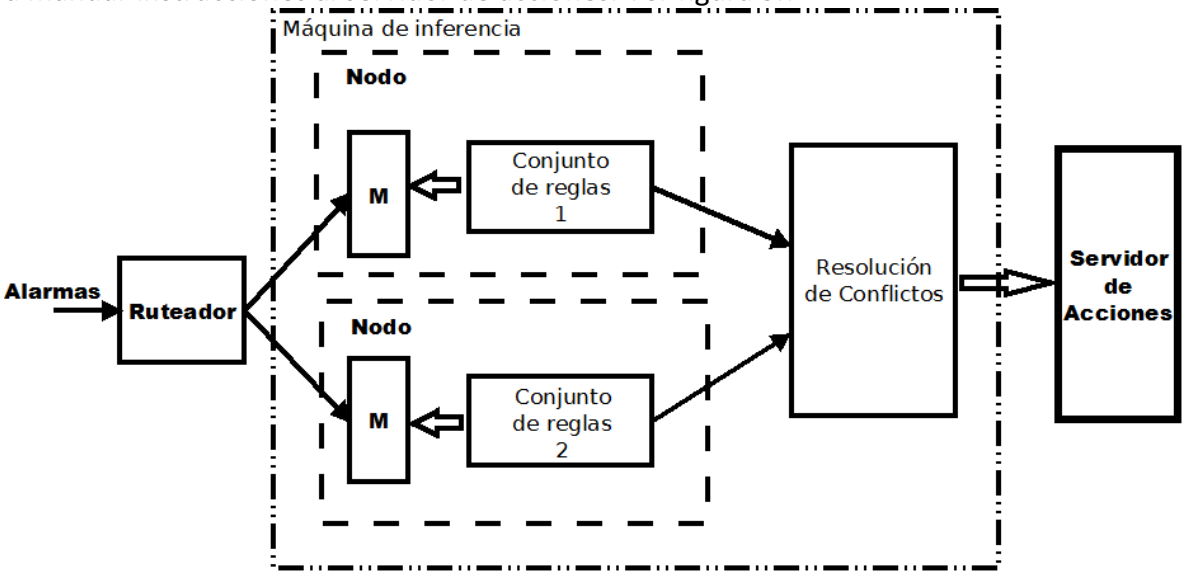

Figura 3.7: Emparejamiento de reglas en paralelo con un mecanismo de ejecución síncrono y memoria distribuida

Emparejamiento de reglas en paralelo con un mecanismo de ejecución asíncrono y memoria distribuida: Aquí se distribuyen las reglas en bloques, duplicando aquellas reglas que no requieren mantener la consistencia, se distribuyen los datos en conjuntos disjuntos para cada procesador, para localizar el procesador que contiene cierta información, se aplica una técnica de hashing. Si durante la etapa de emparejamiento se requiere de una pieza de información residente en otro procesador se manda un mensaje a ese procesador solicitando el dato. La ejecución es

asíncrona ya que las reglas no interferirán entre sí, por lo que pueden ser ejecutadas una vez evaluadas. Ver figura 3.8

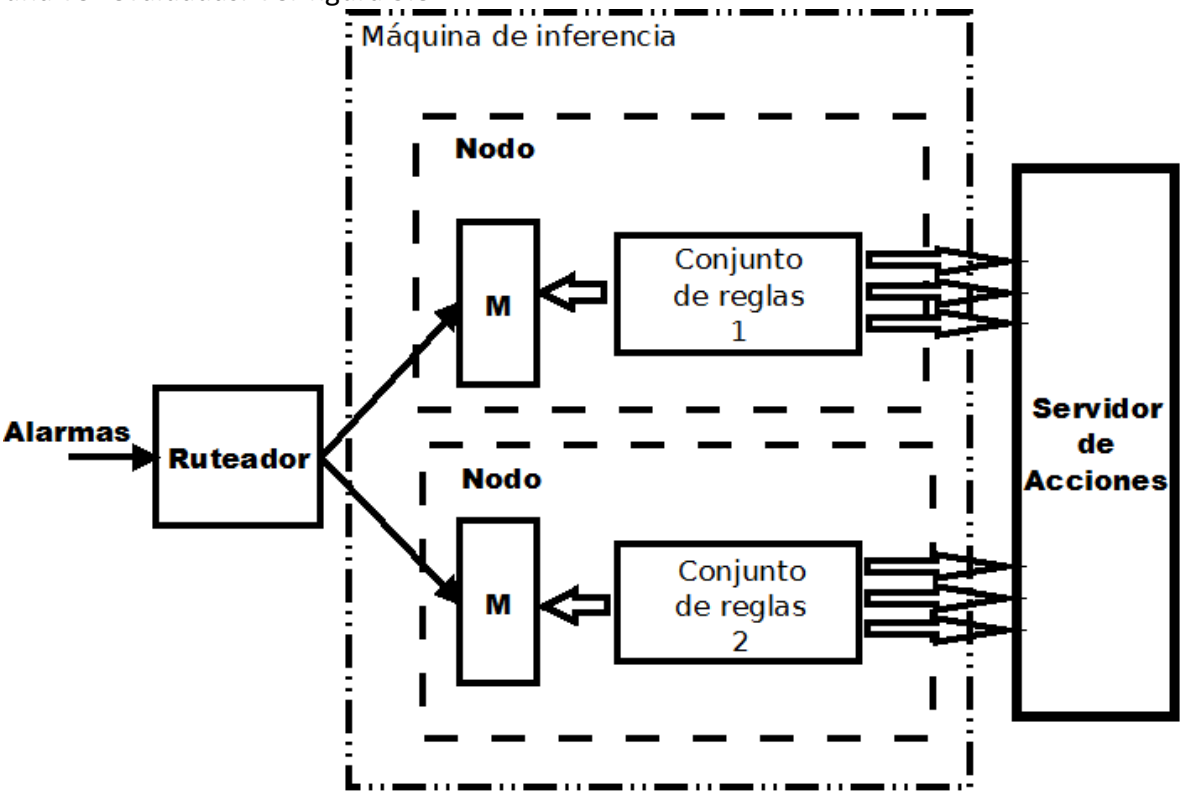

Figura 3.8: Emparejamiento de reglas en paralelo con un mecanismo de ejecución asíncrono y memoria distribuida

Finalmente se optó por la arquitectura con un emparejamiento en paralelo sobre memoria compartida con un mecanismo de ejecución asíncrono, con el fin de evitar la necesidad de ejecutar una comunicación entre procesadores pero al mismo tiempo aprovechar las ventajas de una arquitectura multiprocesador y disminuir el tiempo que tarda una alarma en ser atendida.

Para la etapa de emparejamiento se empleará el mecanismo conocido como encadenamiento hacia adelante, previamente expuesto en la sección 2.2. La elección de este mecanismo se basó en las siguientes razones:

- Se tiene un gran número de hechos contra un número comparativamente menor de reglas. Este tipo de ambientes no es el más adecuado para las técnicas de RETE y TREAT ya que es necesario insertar y borrar nodos en la red constantemente (7).

- La naturaleza del problema, aquí cada evento desencadena una evaluación de 0 a más condiciones, las cuales una vez cumplidas pueden activar otro evento, el cuál ingresa directamente a la memoria de trabajo para ser evaluado por las demás reglas o bien enviado al gestor de fallas.

Se presenta un sencillo ejemplo para ilustrar el mecanismo de razonamiento por encadenamiento hacia adelante a emplear:

R1: “Cuando evento1 entonces crea evento2”
R2: “Cuando evento2 entonces crea evento 3”

Con encadenamiento hacia adelante, al detectarse el evento1, se procederá a la evaluación de las condiciones, en este caso no hay condición que evaluar, por lo que se procede a ejecutar la acción “crea evento2” el evento puede ahora ser detectado por la regla 2, con lo cual prosigue el proceso de razonamiento.

Ahora bien, el proceso de evaluación de las condiciones expuestas en la sección 3.4.2. no es trivial, ya que es necesario realizar procesos de búsqueda en la memoria de trabajo y la topología. Esto es, una búsqueda de la satisfacción de las condiciones en la parte “Si Condición” de la regla. Se opta por una búsqueda secuencial sobre las estructuras de datos y una consulta a la base de datos, dependiendo de la fuente de información a utilizar.

Una vez teniendo en cuenta estas consideraciones, se puede diseñar de manera detallada la arquitectura de la máquina de inferencia, la cual se presenta en la figura 3.9.

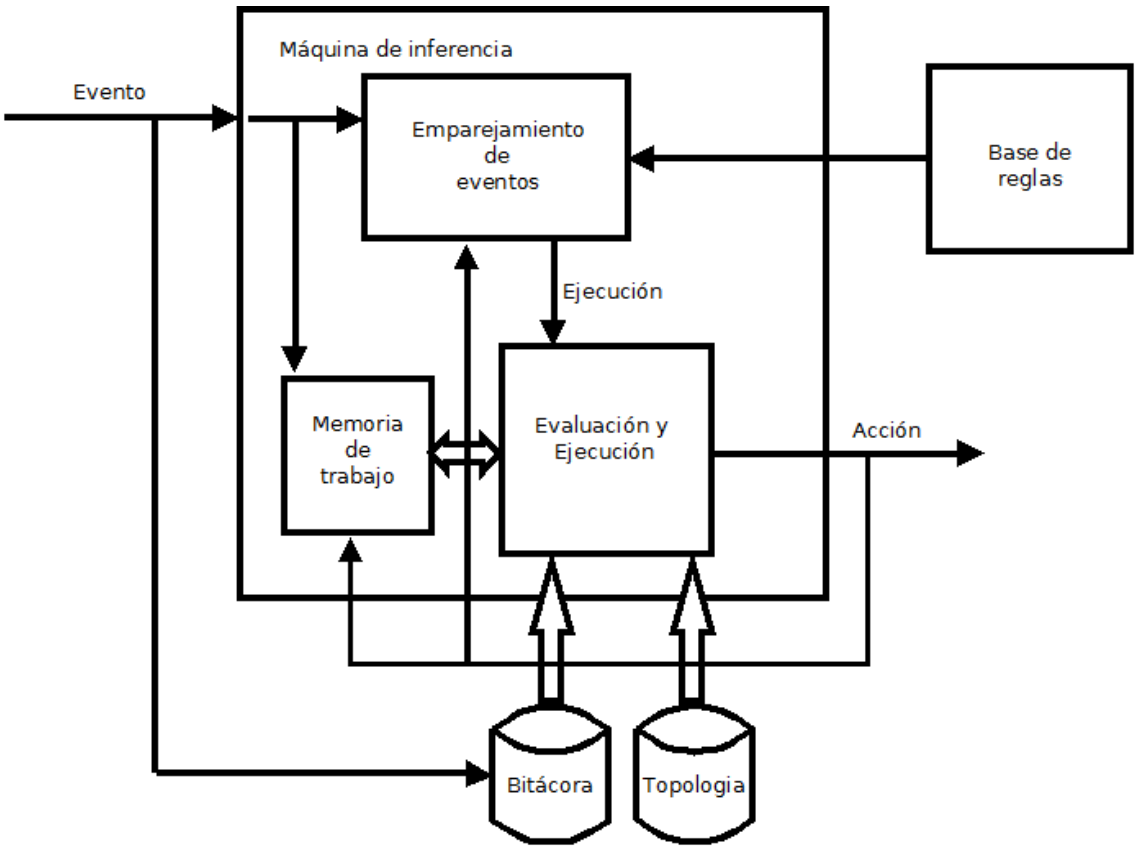

Figura 3.9: Diagrama detallado de la máquina de inferencia con los principales componentes.

Los componentes identificados son los siguientes:

Emparejamiento de eventos: un proceso que se encarga de leer los objetos provenientes del control de mensajes. Empareja el evento recibido contra una lista de "eventos activadores" (la clausula "Cuando evento" de la regla"). En caso de detectarse un evento activador se procede a llamar a la evaluación de condiciones. En caso de no haber condiciones a evaluar se procede a ejecutar una acción.

Evaluación y ejecución: Inicia un proceso para la regla pasando una referencia al evento. Cada regla se encuentra compilada y procederá a realizar una búsqueda en la memoria de trabajo, una búsqueda en la bitácora de eventos detectados y/o búsqueda de relaciones entre objetos en la topología.

Memoria de trabajo: Estructura de datos que contiene un historial de eventos recibidos y cuyo estatus es "activo". Esta estructura de datos debe implementar un mecanismo de "seguros" para evitar interferencias de lectura-escritura entre las reglas.

Bitácora: Tabla en una base de datos que contiene un registro de todos los eventos recibidos. Es una memoria a largo plazo que será utilizada para aquellas acciones de correlación que requieran una búsqueda de eventos que ya no residan en la memoria de trabajo.

Base de reglas: Conjunto de reglas, con definición de eventos activadores y evaluación de condiciones, el emparejador leerá de la base las condiciones de activación para cada regla y procederá a ejecutar su sección de evaluación de condiciones si la regla se da por activada.

A continuación se resume el proceso utilizado la evaluación de una regla:

1. Un objeto de tipo registro de alarma llega a la máquina de inferencia a través del control de mensajes.

2. El objeto es enviado a la memoria de trabajo, mientras, el módulo de emparejamiento de eventos busca eventos activadores de las reglas, (un evento correspondiente a la condición "Cuando evento"). Esto se realiza de manera paralela, es decir el mismo evento es emparejado con una regla diferente en un proceso o hilo diferente.

3. Se identifica el objeto como un evento activador de una regla R.

4. Se comienza un proceso, el cuál contendrá la forma compilada de la regla R. Este proceso corresponderá a una instancia de la regla R.

5. El proceso de evaluación de R se realiza hasta que se llega a una conclusión, la conclusión puede ser "Verdadero" en cuyo caso la clausula "Entonces" de la regla es mandada al servidor de acciones en forma de un mensaje o bien es

mandado a la memoria de trabajo, en este caso se le notifica al emparejador de eventos para que empareje el evento generado en busca de condiciones de activación. Si el resultado es “Falso” el proceso termina.

6. El servidor de acciones tomará la acción correspondiente al mensaje indicado, aplicando políticas de prioridad y resolución de conflictos para la ejecución de las reglas de ser el caso.

Con este proceso se tiene un emparejamiento en paralelo (cada evento es emparejado con cada regla de manera paralela) y una cantidad n de instancias de reglas en memoria principal evaluándose paralelamente, conforme a lo descrito en la arquitectura de memoria compartida con emparejamiento en paralelo y ejecución asíncrona (una vez que termine la evaluación de condiciones se procede a ejecutar la acción), como se tenía pensado.

Se procede definir la forma en que los activadores son seleccionados y las reglas son evaluadas:

- Primero se define la representación de una regla en el sistema. Una regla es un objeto con 2 métodos. El método “Activador” que recibe como parámetro un evento. Este método empareja el evento mediante una serie de condiciones if (más adelante se explicará la forma de codificación de las reglas). Si el evento evalúa verdadero para una o más condiciones entonces se da el evento como reconocido y la regla procede a evaluarse.

- El método “Evaluador” recibe cómo parámetros el evento y una referencia a la memoria de trabajo. Este método constituye el código de evaluación de la regla, la forma en que se genera dicho código se explica más adelante.

Ahora bien, teniendo en cuenta esta base se procedió a definir la regla como una interfaz en una jerarquía orientada a objetos. Cada regla nueva corresponde a una nueva clase que implementa dicha interfaz que contiene los métodos “Activador” y “Evaluador”.

Las Reglas ya compiladas se encuentran almacenadas en una biblioteca dinámica. Esta biblioteca contiene una colección de reglas cuyos métodos “Activador” y “Evaluador” se encuentran ya implementados.

El emparejador de eventos se encarga de recibir el evento de la cola de mensajes o de la memoria de trabajo, luego manda a llamar al método “Activador” de cada una de las reglas pasando cómo parámetro el evento recibido. En caso de que el método “Activador” evalúe cómo verdadero, se procede a comenzar un hilo con el método “Evaluador” de la regla y con el evento como parámetro.

Cabe mencionar que ni el nombre ni los atributos de las reglas son conocidos por el emparejador de eventos en tiempo de compilación. Por lo que fue necesario utilizar la técnica conocida como “reflexión” para poder llamar a los métodos de las reglas.

La “reflexión” consiste en “la capacidad de un programa para manejar como datos una representación del estado del programa durante su ejecución” (28), esta noción fue propuesta en (29). En este caso se utilizó la reflexión para poder llamar a los métodos de una clase (incluyendo el constructor) e instanciarla sin conocer su nombre en tiempo de compilación. Esto permite que las reglas y el sistema trabajen en manera conjunta sin que sea necesario recompilar todo el sistema cada vez que se realice una modificación a la base de reglas.

3.4 Generación de reglas

Ahora se revisa el procedimiento de generación y compilación de reglas para ser usadas por la máquina de inferencia.

Para la creación de las reglas se utilizará un editor de reglas, que proporcionará una interfaz de usuario mediante la cual el operador del sistema experto podrá crear y editar las reglas en la base de reglas.

Las reglas generadas estarán en forma de un archivo xml, este archivo contendrá las reglas en un formato unificado para que puedan ser compiladas al lenguaje de programación que sea necesario.

A partir de estas reglas se utilizará un algoritmo de generación de código para obtener un código fuente, dicho código fuente se compilará para crear una biblioteca dinámica de clases. Esta biblioteca será cargada por la máquina de inferencia en tiempo de ejecución, lo cual permite trabajar con una base de reglas independiente a la máquina de inferencia.

Para facilitar el proceso de compilación de códigos se realizaron un conjunto de funciones de apoyo para el proceso de evaluación de condiciones, dichas funciones se encuentran en otra biblioteca, por lo cual una referencia a dicha biblioteca debe de ser insertada en la base de reglas.

La creación del editor de reglas y del generador de código está más allá del alcance de este trabajo, pero se va a especificar la base para la creación de dicho código.

3.4.1 Clase Regla

Cada regla en la biblioteca de clases está representada por una clase, dicha clase implementa una interfaz predeterminada. Dicha interfaz contiene el prototipo para los métodos “Activador” encargado de emparejar un evento en busca de la activación de una regla. El otro método es “Evaluador” el cuál se encarga de evaluar las condiciones de la regla una vez que el activador ha evaluado como verdadero.

Cada regla puede contener sus propios atributos y métodos, ya que el único elemento necesario para su compatibilidad con la máquina de inferencia es la implementación de los métodos “Activador” y “Evaluador”.

A su vez, la máquina de inferencia crea una instancia de la regla cada vez que se evalúa un activador de la clase. Por lo que en un determinado momento se pueden tener 1 o más instancias de la regla evaluándose paralelamente. En la figura 3.10 se aprecia el diagrama de clases para la implementación de la interfaz “Regla”.

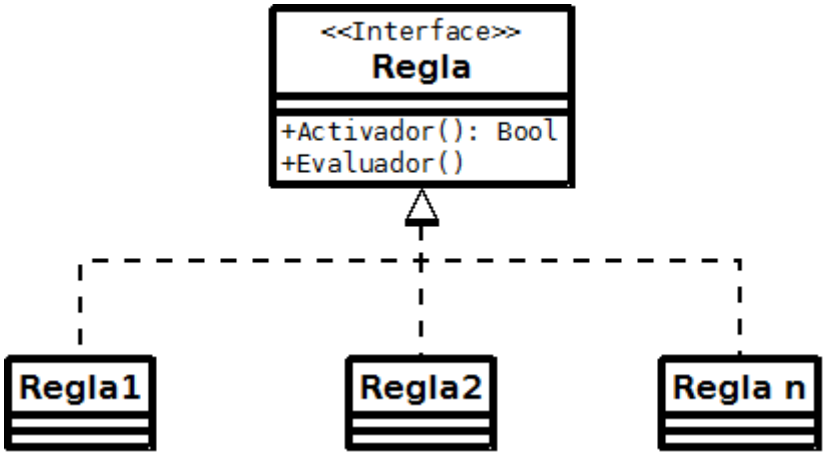

Figura 3.10: Diagrama de clases para la implementación de la interfaz Regla por una o más reglas.

3.4.2 Evaluación de eventos activadores.

La evaluación de un evento activador consiste en comparar los atributos de un evento en busca de valores determinados por el usuario. En el caso del código generado, el evento es un registro de alarma. El registro de alarma viene definido por un dato de tipo estructura (struct) cuyos miembros corresponden a los de la tabla 3.1., dicha estructura se presenta a continuación.

```
public struct RegistroDeAlarma
    {
        public string TipoDeSystema;
        public int TipoDeRegistro;
        public int IdRegistro;
        public string Objeto;
        public int Severidad;
        public DateTime Hora;
        public int TipoDeEvento;
        public int Causa;
        public int ProblemaEspecifico;
        public string Texto;
        public int NumeroDeAlarma;
        public int CategoriaDeAlarma;
        public int ClaseDeAlarma;
        public int Status;
```

```
        public string SolucionPropuesta;

    }
```

El parser de la interfaz entre el manejador de fallas y la máquina de inferencias "traduce" los mensajes de alarma al formato de la estructura asignando los valores de los campos correspondientes a los contenidos en el mensaje.

El método "Activador" evaluará los campos indicados por la regla en busca de satisfacer ciertas condiciones:

- En caso de enteros se evaluarán los operadores relacionales <,>,==,<=,>=. Se comparará el valor del campo para el evento contra un valor determinado. Además de las operaciones suma, resta, división y multiplicación en caso de ser necesario.

- En caso de cadenas de caracteres se utilizarán las operaciones "Contiene" para buscar una subcadena dentro de la cadena, e "Igual" para evaluar si la cadena es igual a una cadena dada.

- En caso de los valores de tipo DateTime, se realizan las operaciones de comparación (>, < correspondientes a "antes" y "después" de una fecha determinada).

Es importante destacar que la evaluación de los eventos activadores es una operación que debe mantenerse relativamente simple, ya que sólo busca ciertas características del evento, sin compararlo con otros eventos ni con información previa del estado del sistema. El propósito del método "Evaluador" es realizar dichas evaluaciones. Por lo que el método "Activador" puede constituirse a partir de expresiones "if". En caso de tener más de una condición se genera código de la forma:

If(RegistroDeAlarma.[Miembro1]>[valor] And
RegistroDeAlarma.[Miembro2]>[valor2]….)

Ya que sólo se busca que un evento cumpla con características determinadas, sólo se evalúa una vez el valor de cada campo de la estructura.

Es de notarse que pueden existir dos reglas diferentes con métodos "Activador" iguales, esto es, un mismo evento puede disparar 1 o más reglas. Esto con el fin de permitir la evaluación paralela de diferentes condiciones y realizar diferentes procedimientos de correlación y atención a fallas a un mismo evento.

Finalmente se trata el requisito de los eventos "reconocido" y "atendido". En el caso de "reconocido" simplemente se evalúa el valor del campo "status" del registro de alarma. Ya que los únicos cambios permitidos en la memoria de trabajo son "añadir" y "borrar", la actualización de un registro se realiza mediante las acciones "borrar" del registro actual en la memoria de trabajo, y se manda una copia exacta del evento al

gestor de fallas o a la memoria de trabajo. Esta copia tiene la única diferencia en que el valor del campo “status” ha cambiado para reflejar su estado de “reconocido”. Caso similar con “atendido”, sólo que en este caso los eventos “atendido” no son almacenados en la memoria de trabajo, más si son almacenados en la bitácora.

3.4.3 Evaluación de condiciones

Como se mencionó en la sección 3.2.4 existen 3 tipos de condiciones a evaluar una vez que se ha detectado un evento activador:

- Buscar en la memoria de trabajo o en la bitácora de alarmas la existencia de otro evento con ciertas características

- Ver si el objeto cumple con cierta relación con otro objeto dentro de la topología.

Para el primer punto, cuando se tiene que buscar en la memoria de trabajo por un evento activo que cumpla ciertas características, se tienen en cuenta las siguientes consideraciones:

- En la memoria de trabajo se encuentran los eventos activos (cuyo atributo “Estatus” tiene valores 1 ó 2).

- Además existe una bitácora de todas las alarmas recibidas por el sistema experto, está bitácora se encuentra en una tabla en una base de datos.

- La regla debe buscar entre los elementos activos o incluso entre los elementos de la bitácora, dependiendo del criterio del experto.

La búsqueda en la memoria de trabajo y en la base de datos, de manera semejante a la evaluación de eventos activadores, permite las siguientes condiciones evaluadoras:

- En caso de enteros se evaluarán los operadores relacionales <,>,==,<=,>=.

- En caso de cadenas de caracteres las funciones “Contiene” e “Igual”

- En caso de los valores de tipo DateTime, se realizan las operaciones de comparación y restas entre fechas.

- Uso de variables de tipo contadores para evaluar el número de ocurrencias de cierto evento.

- Umbrales de tiempo, para evaluar si un evento sucedió antes o después de otro evento.

En la solución propuesta en este trabajo, las variables se van a implementar como miembros de la clase Regla. La generación de esta clase corre a cargo del sistema editor de reglas, como se ha mencionado anteriormente. Estos miembros serán privados, ya que la arquitectura no contempla un mecanismo de intercambio de información entre reglas aparte de la emisión de mensajes en la etapa de "Acción". Debido a esto la inicialización de los valores de las variables debe de realizarse en el constructor default de la clase (sin parámetros).

Teniendo estas consideraciones se procede a definir la forma de realizar dichas operaciones. Las búsquedas en la memoria de trabajo se realizan de manera secuencial. Una función se encarga de comparar cada elemento de la memoria de trabajo con los valores indicados en las reglas a través de sentencias if en forma análoga a la evaluación de eventos activadores, sólo que añadiendo el uso de contadores y umbrales, por lo que es necesario añadir éstas dos operaciones:

- Incrementar contador.
- Verificar umbral de tiempo.

En caso del código generado, estas operaciones se traducen en:

- Contador=Contador+1.
- Si Fecha1-Fecha2 <= Umbral.

Siendo Contador y Umbral miembros de una clase de tipo Regla. Como cada instancia de regla es independiente (corresponde a un objeto en memoria) el valor de las variables es independiente en cada regla.

En el caso de que la búsqueda se realice sobre la bitácora de alarmas en la base de datos, será necesario generar una sentencia SQL del tipo "Select [atributo] from Bitácora where [condición]". Donde condición corresponde a las sentencias "if" en el caso detallado anteriormente, por lo que en el generador de código, solamente será sustituido el valor de [atributo] por el atributo que se desea comparar y adicionalmente se genera la secuencia condicional de manera análoga a como se genera una secuencia condicional para búsquedas en la memoria de trabajo, sustituyendo el valor de [condición] en la sentencia SQL. Una vez realizada la consulta, si esta devuelve mínimo un registro, la condición se da como cumplida ("verdadera").

Finalmente queda el tercer punto, la búsqueda de relaciones entre objetos, las relaciones sólo serán de tres tipos, como se expuso en la sección 3.2.1, estos tipos de relación son:

- Contención.
- Herencia.

- Conexión.

Con el motivo de verificar dichas condiciones se realizaron 3 funciones, las cuales han sido compiladas a una biblioteca dinámica que debe de ser referenciada por cada Regla. Dichas funciones exploraran el archivo xml en busca de la relación solicitada entre los objetos indicados, en caso de encontrarla retornarán un valor de "verdadero", por lo que se da la condición como cumplida.

Las funciones son:

Contiene(string objeto1,string objeto2) : Retorna un valor booleano "verdadero" si el objeto1 contiene al objeto 2 en la jerarquía de objetos de la topología.

Hereda(string objeto1, string objeto2): Retorna un valor booleano de "verdadero" si el objeto1 es de tipo objeto2, es decir hereda de objeto2.

Conexión(string objeto1,string objeto2): Retorna un valor de "verdadero" si el objeto1 y el objeto2 forman parte de un "link" dentro de la topología.

3.4.4 Acciones

Una vez que se han evaluado como verdaderas las condiciones de una regla, se puede proceder a la etapa de ejecución de acciones, a si como funciones compiladas en una biblioteca dinámica, estas acciones son:

- Envío de un evento al gestor de fallas, dicho evento debe de estar definido en la regla, básicamente consiste en una instancia de la estructura registro de alarmas, esta instancia es codificada a forma de cadenas de caracteres a través de otro objeto parser, el cuál realiza la operación inversa al parser de la interfaz entre la máquina de inferencia y el gestor de fallas. Una vez codificado en forma de cadena de caracteres el mensaje se envía al gestor de fallas para que entre en el proceso de gestión de fallas. Se desarrolló la función EnvioEvento para desempeñar esta tarea.

- Envío de un evento a la memoria de trabajo, en lugar de enviar un elemento al gestor de fallas, se envía directamente a la memoria de trabajo, esto permite proseguir con el proceso de inferencia mediante encadenamiento hacia adelante. No se envía una copia al gestor de fallas ya que podría generar una duplicación de mensajes (el mensaje es enviado al gestor de fallas y el gestor de fallas lo envía al sistema experto). La memoria de trabajo contiene un método especial para alertar cuando un evento ha sido agregado por una regla. Se creó la función EnvioMemoria que efectúa el proceso mencionado.

- Envío de un mensaje al servidor de acciones. Este mensaje puede ser desde una alerta, una nota, un recordatorio o un comando, el tipo de mensajes recibidos y

acciones a tomar por el servidor de acciones están fuera del alcance del presente trabajo. La función EnvioMensaje se encarga de cumplir este objetivo.

Una vez concluidas las consideraciones generales de diseño se procedió a desarrollar el sistema, haciendo las adaptaciones pertinentes a la plataforma de desarrollo y el ambiente de trabajo.

4 Desarrollo de la máquina de inferencia

A continuación se describe brevemente el desarrollo y construcción de los componentes de la máquina de inferencia. La plataforma de desarrollo es Microsoft .Net con el lenguaje de programación Visual C#, servidor MSQM y SQL Server 2010 express como sistema manejador de base de datos.

El sistema se implementó en una arquitectura orientada a objetos, las dependencias entre las clases que conforman al sistema se aprecian en la figura 4.1.

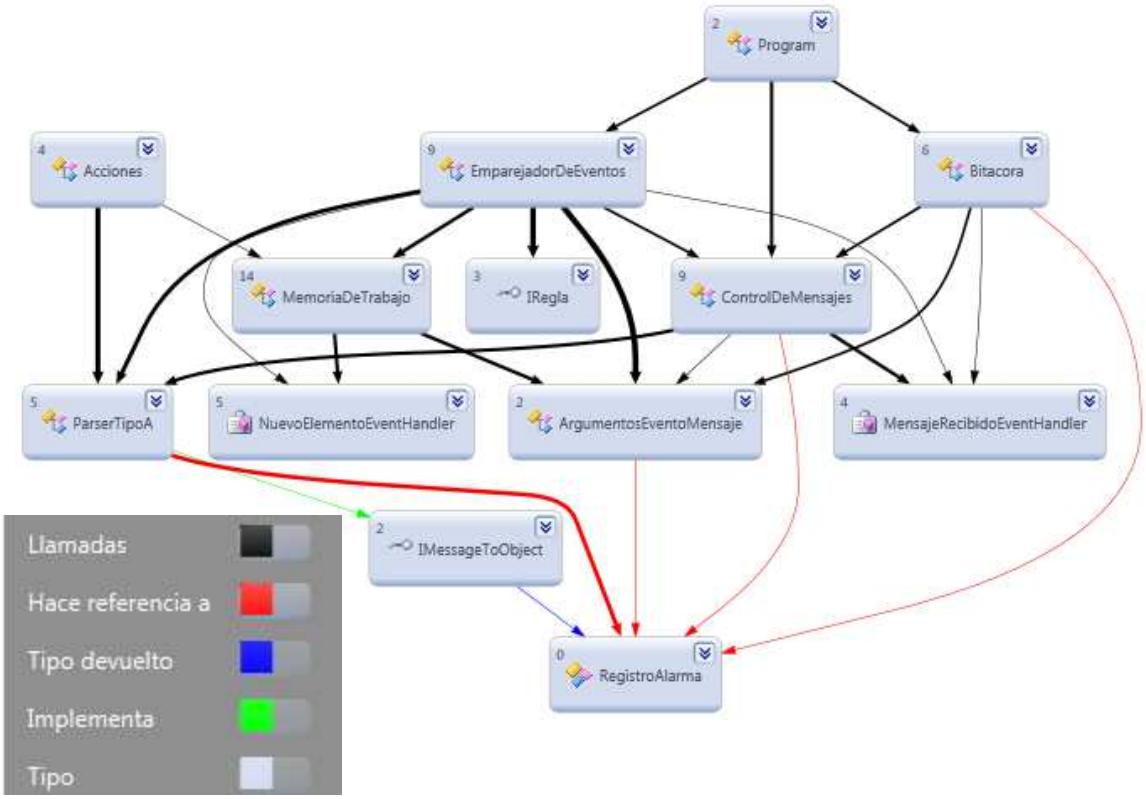

Figura 4.1: Gráfico de dependencias entre las clases del sistema.

A continuación se hace una breve descripción de las clases componentes del sistema así como una explicación concisa de las dependencias mostradas en el gráfico. Posteriormente se presentará el diagrama de clases de los elementos principales.

- Program: Es el punto de entrada de la aplicación, crea dos procesos, uno para el emparejador de eventos y otro para el control de mensajes, además crea una instancia de la clase Bitacora, la cual servirá para almacenar los mensajes recibidos en una tabla en la base de datos.

- ControlDeMensajes: Lee los mensajes de la cola, manda a llamar al parser (en este caos ParserTipoA) para traducir el mensaje. Utilizando la instancia de la clase Bitacora almacena el mensaje en la base de datos. Finalmente envía un evento de tipo “MensajeRecibido” a través del

delegado “MensajeRecibidoEventHandler”, este evento será recibido por el EmparejadorDeEventos.

- MensajeRecibidoEventHandler: El modelo de eventos de .NET Framework se basa en la existencia de un delegado de eventos que conecte un evento a su controlador. Para provocar un evento, se requieren dos elementos:

    - Delegado que identifica el método que proporciona la respuesta al evento.

    - Clase que contiene los datos de eventos.

  El delegado es un tipo que define una firma, es decir, el tipo del valor devuelto y los tipos de lista de parámetros de un método. Se puede utilizar el tipo de delegado para declarar una variable que puede hacer referencia a cualquier método con la misma firma que el delegado (30).

  La firma estándar de un delegado del controlador de eventos define un método que no devuelve ningún valor, cuyo primer parámetro es del tipo Object y hace referencia a la instancia que provoca el evento y cuyo segundo parámetro se deriva del tipo EventArgs y contiene los datos de evento. Si el evento no genera los datos de evento, el segundo parámetro simplemente es una instancia de EventArgs. De lo contrario, el segundo parámetro es un tipo personalizado derivado de EventArgs y proporciona los campos o las propiedades necesarias para contener los datos de evento (30).

  EventHandler es un delegado predefinido que representa específicamente un método controlador para un evento que no genera datos. Si su evento genera datos, debe suministrar su propio tipo de datos de evento personalizado y crear un delegado donde el tipo del segundo parámetro sea el tipo personalizado o utilizar la clase de delegado genérico EventHandler y sustituir el tipo personalizado por el parámetro de tipo genérico (30).

  En este caso MensajeRecibidoEventHandler es un delegado que genera datos de tipo “RegistroAlarma”, dichos datos están definidos en la clase ArgumentosEventoMensaje.

- ArgumentosEventosMensaje: Tipo de datos de evento, el cual consiste en un registro de alarma como está definido en la clase “RegistroAlarma”.

- EmparejadorDeEventos: Clase que recibe un registro de alarma del control de mensajes a través del delegado MensajeRecibidoEventHandler, lo guarda en la memoria de trabajo y procede a realizar el emparejamiento de eventos a través del método “Activador” encontrado en las reglas (en el diagrama representado por la interface “IRegla”). En caso de que el método “Activador” se evalúe verdadero se realizará la evaluación de las condiciones a través del método “Evaluador” de la regla activada.

- MemoriaDeTrabajo: Estructura de datos que contiene los registros de alarma activos. Manda a llamar al delegado NuevoElementoEventHandler cada vez que se añade un nuevo elemento a la memoria a través del método “Evaluador” de una regla.

- NuevoElementoEventHandler: Delegado que genera datos de tipo “RegistroAlarma” cuando se añade un elemento a la memoria de trabajo a través del método “Evaluador” de una regla.

- Acciones: Clase que se encarga de realizar las acciones indicadas por el método “Evaluador” de las reglas. Interactúa con la MemoriaDeTrabajo al añadir o eliminar mensajes y con el “Parser” para transformar los registros de alarma a un formato de texto (la operación inversa a la traducción realizada por el control de mensajes).

- ParserTipoA: Clase que se encarga de la traducción de mensajes de formato texto a formato “RegistroAlarma” y viceversa. En este caso el parser fue diseñado para el formato de registro de alarma definido con anterioridad. Cabe destacar que es la implementación de la interfaz IMessageToObject, por lo cual si es necesario reemplazar el parser para utilizar un formato de registro distinto, simplemente será necesario implementar dicha interfaz en el nuevo parser.

- IMessageToObject: Interfaz que determina la forma en que debe estar conformado un parser, permitiendo separar la implementación del parser del resto del funcionamiento del sistema.

- RegistroDeAlarma: Tipo de datos fundamental a ser utilizado por la máquina de inferencia, como está definido en la sección 3.4.2.

A continuación se presenta el diagrama de clases de los elementos más importantes de la máquina de inferencia.

4.1. Interfaz entre el gestor de fallas y la máquina de inferencia

Se presenta el diagrama de clases del módulo de interfaz entre el gestor de fallas y la máquina de inferencia en la figura 4.2.

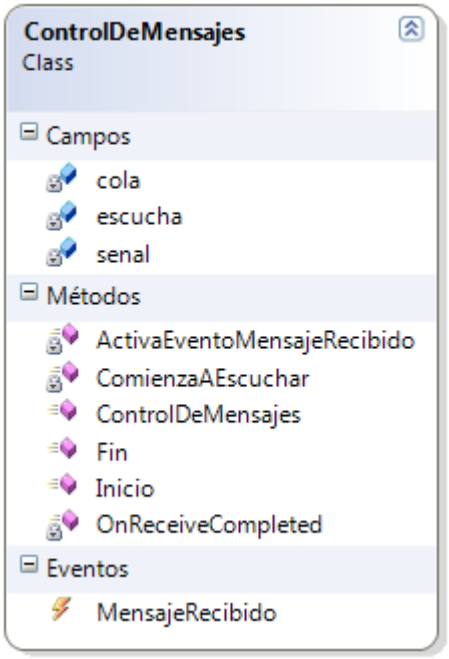

Figura 4.2: Diagrama de clases para la interfaz entre el gestor de fallas y la máquina de inferencia

A partir de la clase ControlDeMensajes se instanciará un objeto que inicia un proceso que se mantiene en ejecución al momento de ejecutar el método Inicio() y culmina cuando se ejecuta el método Fin(). Los campos son:

escucha: Es una bandera, mientras la bandera sea verdadera el proceso continuará, cuando sea falsa el proceso termina.

cola: una referencia a la cola de mensajes provenientes del gestor de fallas, esta referencia es proporcionada por un servidor de colas, dicho servidor puede ser implementado o acoplado a partir de uno ya existente, como por ejemplo MSQM o JMS.

senal: señal de sincronización del proceso, cada vez que se lee un mensaje se manda una señal de espera por el próximo mensaje.

Los métodos son:

ControlDeMensajes: Constructor de la clase, recibe como parámetro la referencia a la cola de mensajes y lo guarda en el campo “cola”.

Inicio: Método que comienza la escucha de mensajes, asigna a la bandera “escucha” el valor de verdadero, y prepara la recepción de mensajes de la cola. Finalmente llama al método ComienzaAEscuchar que es el encargado de recibir mensajes de la cola.

Fin: A través de este método se le indica al proceso que termine su ejecución.

ComienzaAEscuchar: Realiza la recepción de mensajes de la cola, invocando al método BeginReceive de la cola representada por “cola”, este es un método de recepción asíncrono, en cuanto haya un mensaje en la cola y sea posible efectuar una lectura, esta se efectuará, en tanto no se realiza acción alguna. Después de invocado este método le indica al proceso que espere por el siguiente mensaje a través de la variable signal, si no se utilizara esta variable el programa terminaría al realizar la primera recepción, ya que considera su acción ejecutada.

OnReceiveCompleted: Al leerse un mensaje, se invoca a este método, el cual ejecuta las acciones. Esto se lleva a cabo de la siguiente manera: en el método ComienzaAEscuchar se invoca el método BeginReceive de la cola de mensajes proporcionada por el servidor de colas, el método genera un evento de tipo ReceiveCompleted, esto ocurrirá cuando se lea un mensaje de la cola. Este evento invoca al método OnReceiveCompleted.

Este método primero le indica a la cola que deje de recibir mensajes en lo que procesa el mensaje actual. Luego crea una instancia del “Parser” para traducir el mensaje a un formato de objeto (Registro de Alarma). Finalmente manda a llamar al delegado MensajeRecibido.

MensajeRecibido: Este delegado conforma la comunicación existente entre la interfaz y el emparejador de eventos, ya que el emparejador de eventos recibirá el registro de alarmas como argumento a través de este delegado de tipo MensajeRecibidoEventHandler.

Un delegado es similar a un puntero a funciones en C++, éste permite encapsular la referencia a un método dentro del delegado. El delegado puede ejecutar el método referenciado sin saber en tiempo de compilación el nombre del método.

En la plataforma de trabajo utilizada, un evento del programa (no confundir con un evento del sistema de alarmas) ocurre cuando una clase “publicador” publica un mensaje (invoca un método a través de un delegado). El “publicador” puede tener uno o varios “subscriptores”. Cuando se publica un mensaje todos los “subscriptores” son notificados. Cada subscriptor a su vez puede ejecutar un método a través del delegado. El método a ejecutar es definido por el “subscriptor”.

La figura 4.3 muestra una forma simplificada del diagrama de interacción entre la cola de mensajes, el control de mensajes y el emparejador de eventos. Como se aprecia, los eventos OnReceiveCompleted() y MensajeRecibido() son el medio de comunicación entre la cola de mensajes, el control de mensajes y el emparejador de eventos. El

Control de Mensajes es un subscriptor de la Cola de mensajes y el Emparejador de eventos es a su vez un subscriptor del Control de Mensajes.

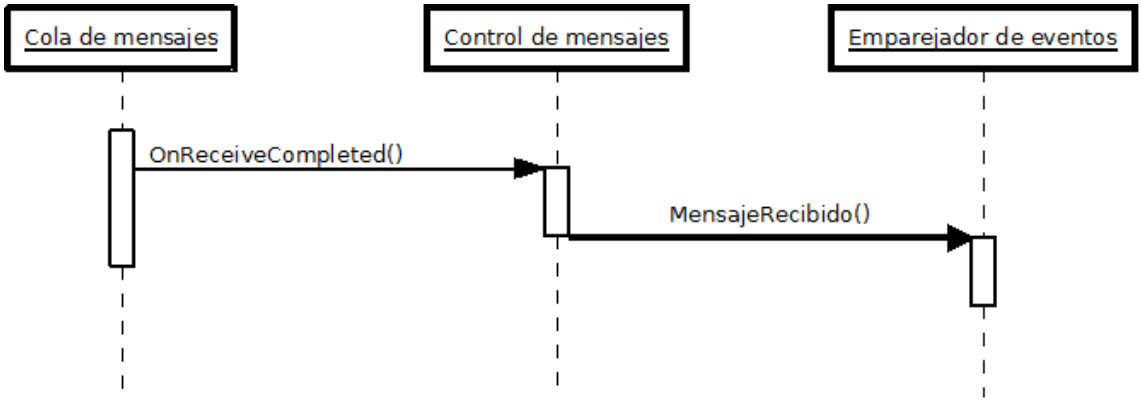

Figura 4.3: Diagrama de dependencia entre la cola de mensajes, el control de mensajes y el emparejador de eventos.

4.2. Emparejador de eventos

La figura 4.4 presenta el diagrama de clases del emparejador de eventos.

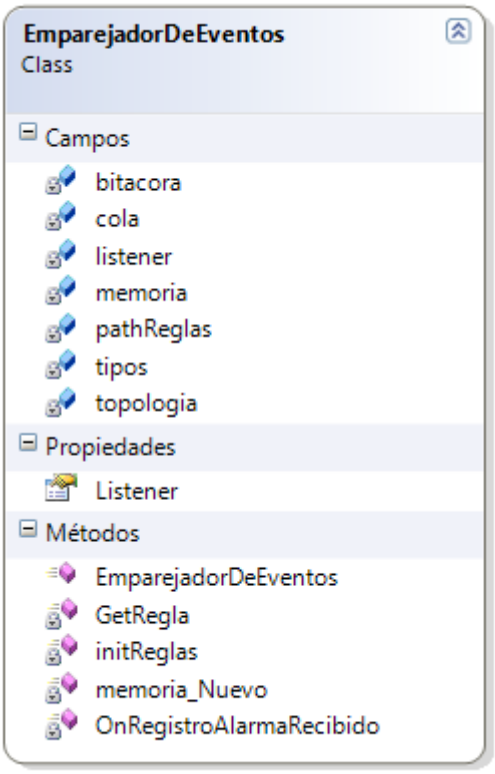

Figura 4.4: Diagrama de clases del emparejador de eventos.

El emparejador de eventos tiene los siguientes campos:

listener: Una instancia de ControlDeMensajes, es la referencia al servidor que recibirá mensajes de la cola y los traduce a un formato RegistroAlarma. La propiedad Listener consiste en la encapsulación del campo listener con métodos set() y get().

memoria: Una referencia a la memoria de trabajo a utilizar.

pathReglas: Una cadena que almacena la dirección de la base de reglas dentro del árbol de directorios del sistema, tener en cuenta que la base de reglas es una biblioteca dinámica.

tipos: Un arreglo de objetos de la clase Type (perteneciente al lenguaje C#, contiene la información de las clases contenidas en un ensamblado), mediante reflexión se carga la información de las reglas de la base (en este caso clases de la biblioteca dinámica) y se almacena en este arreglo, para luego poder instanciar cada clase en el proceso de emparejamiento de eventos.

Los métodos de la clase son:

initReglas: Método que se encarga de inicializar la base de reglas almacenando las clases en el arreglo tipos. En sí cada tipo es una estructura de datos que contiene el nombre de la clase y la información de campos y métodos de la misma, para que así puedan ser instanciadas.

GetRegla: Método que recibe como parámetro un objeto Tipo, utilizando la información contenida en el objeto crea un instancia de la regla y regresa una referencia a dicha regla ya instanciada.

EmparejadorDeEventos: Constructor de la clase, recibe como parámetros una referencia al control de mensajes, la asigna a la variable listener, además crea una instancia de la memoria de trabajo y manda llamar a initRegla para inicializar la base de reglas.

OnRegistroAlarmaRecibido: Método que recibe un evento a través del delegado MensajeRecibido, agrega el evento recibido a la memoria de trabajo, luego procede a realizar el emparejamiento del evento recibido con los eventos activadores de las reglas a través del siguiente algoritmo:

```
Para cada clase dentro del arreglo tipos
     Obtener una instancia de la regla a través de
     reflexión
     Si regla.Activador == verdadero
          regla.Evaluador(Mensaje,Memoria)
     Fin Si
Fin Para
```

Ahora bien, el ciclo “para cada” se puede realizar de forma paralela, ya que cada regla se activa de manera independiente. Es decir un proceso o hilo puede leer una regla de

la base de reglas y evaluarlo utilizando el método Activador, por lo que se tiene un emparejamiento paralelo de reglas.

El método “Evaluador” a su vez es independiente del “Activador” una vez que la activación ha sido reconocida como verdadera, este método se puede ejecutar en un nuevo proceso o hilo, el cuál es iniciado. Por lo tanto el proceso puede seguir en memoria aún cuando se entre a un nuevo ciclo de emparejamiento de eventos.

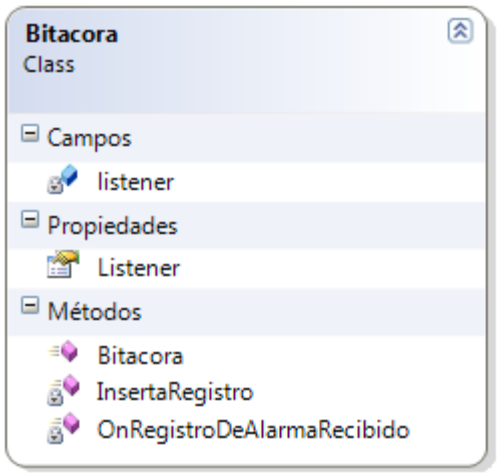

Figura 4.5: Diagrama de clases de la clase Bitacora

En la figura 4.5 se presenta el diagrama de clases de la clase bitácora. Esta clase Bitácora se encarga de guardar los eventos en la bitácora de eventos, la cual es una tabla en una base de datos. Los campos de la clase son:

listener: una referencia al control de mensajes, para recibir la notificación de mensaje recibido de la cola de mensajes.

Los métodos de la clase son:

Bitacora: el constructor de la clase, recibe la referencia al servicio de control de mensajes como parámetro y se prepara para recibir notificaciones de eventos por parte del mismo.

OnRegistroAlarmaRecibido: Método que se ejecuta al recibir un registro de alarma del servicio de control de mensajes a través del delegado MensajeRecibido. Una vez recibido, comienza un hilo con el método InsertaRegistro.

InsertaRegistro: Método que recibe un registro de alarma como parámetro e inserta sus valores en la bitácora. La inserción se hace a través de un procedimiento almacenado en la base de datos, con el fin de evitar declarar explícitamente las consultas en el código del emparejador de eventos, separando la lógica de la base de datos de la lógica del emparejador. Sin embargo el punto común es el formato del

registro de alarmas a almacenar en la tabla de la base de datos, por lo cual dependiendo del registro de alarma el procedimiento almacenado puede cambiar su estructura, modificando los campos de la consulta. En este caso el procedimiento almacenado es el siguiente:

```
CREATE PROCEDURE dbo.usp_insert_message (
         @TipoDeSistema varchar(50),
        @TipoDeRegistro int,
        @IdRegistro int,
        @Objeto varchar(50),
        @Severidad int,
        @Hora datetime,
        @TipoDeEvento int,
        @Causa int,
        @ProblemaEspecifico int,
        @Texto text,
        @NumeroDeAlarma int,
        @CategoriaDeAlarma int,
        @Status int,
        @SolucionPropuesta varchar(50)
)
AS
BEGIN
     INSERT INTO Bitacora
(TipoDeSistema,TipoDeRegistro,IdRegistro,

Objeto,Severidad,Hora,TipoDeEvento,Causa,ProblemaEspecifico
,

Texto,NumeroDeAlarma,CategoriaDeAlarma,Status,SolucionPropu
esta)
     VALUES (@TipoDeSistema,
        @TipoDeRegistro,
        @IdRegistro,
        @Objeto,
        @Severidad,
        @Hora,
        @TipoDeEvento,
        @Causa,
        @ProblemaEspecifico,
        @Texto,
        @NumeroDeAlarma,
        @CategoriaDeAlarma,
        @Status,
        @SolucionPropuesta)

END
```

El método se encarga de llamar al procedimiento y de entregarle como parámetros los campos del registro de alarma. Este método se ejecutará en un hilo diferente, para evitar que el sistema espere a que se dé cómo concluida la operación de escritura para continuar el ciclo de emparejamiento.

4.3 Memoria de trabajo

La Memoria de trabajo consiste en una lista, la lista permite añadir y eliminar elementos, pero anexo a la estructura de datos existe un mecanismo de sincronización. Dicho mecanismo "bloquea" las operaciones de lectura y escritura en la lista, así podemos prevenir problemas de interferencia lectura-escritura y escritura-escritura presentados anteriormente. En la figura 4.6 se presenta el diagrama de clase de la memoria de trabajo.

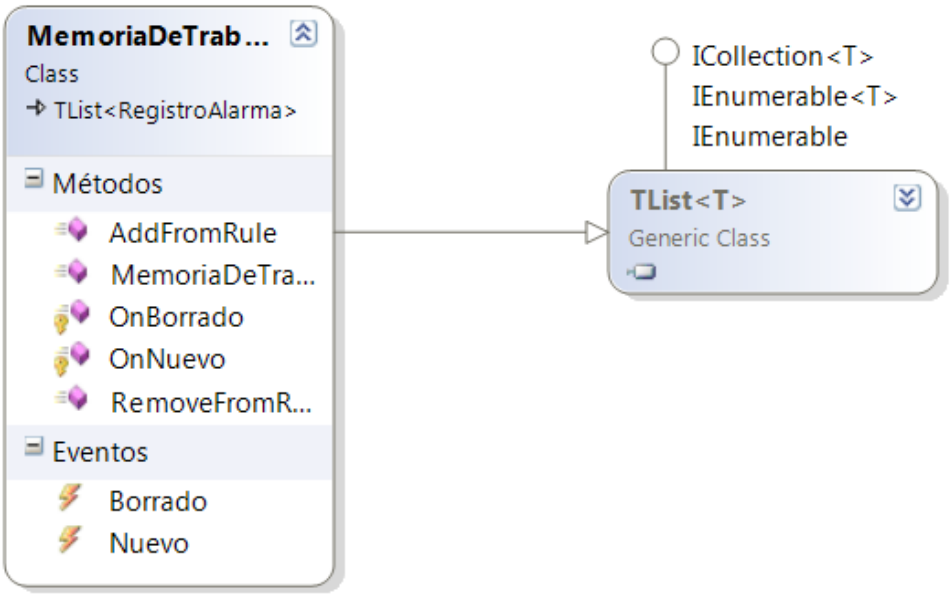

Figura 4.6: Diagrama de clases de la memoria de trabajo.

En el diagrama de clases se aprecia que la memoria de trabajo hereda de la clase genérica TList. TList es una lista sincronizada (con seguros o "locks"), siendo la memoria de trabajo una especificación de dicha clase, a la cuál además de los métodos de la superclase (Add y Remove para añadir y eliminar elementos de la lista) se le han añadido métodos auxiliares para reconocimiento de eventos, los cuales son:

AddFromRule: Método que manda a llamar al método OnNuevo cada vez que se añade un elemento a la memoria de trabajo como acción de una regla.

OnNuevo: Emite un evento de tipo "Nuevo", es decir manda a llamar a un delegado. Este método tiene el fin de que el emparejador de eventos pueda recibir eventos tanto del servicio de control de mensajes como de la memoria de trabajo.

RemoveFromRule: Método que manda llamar al método OnBorrado cada vez que se elimina un elemento de la memoria de trabajo como acción de una regla.

OnBorrado: Emite un evento de tipo "Borrado", es decir manda a llamar a un delegado. Este método tiene el fin de notificar que un evento ha sido eliminado de la memoria de trabajo.

Borrado y Nuevo son delegados que se encargan de notificar al emparejador de eventos cuando se ha borrado o añadido un elemento en la memoria de trabajo respectivamente.

4.4 Simulador

Con el fin de probar el funcionamiento de la máquina de inferencia se desarrolló un pequeño simulador de carga. La principal función de este simulador es generar una serie de mensajes en formato de registro de alarma, estos mensajes serán enviados al servidor de cola de mensajes. De momento se implementará un mecanismo de entrega regular, es decir los mensajes se entregarán en intervalos de tiempo regulares.

Para tener un control de los mensajes generados y enviados, el primer paso en el proceso de simulación consiste en generar los registros de alarma, esto puede ser de manera aleatoria. Los mensajes son generados y guardados en un archivo de texto, o bien se pueden crear de manera manual en el archivo.

Una vez obtenido el archivo de alarmas, un mecanismo de productor consumidor se encarga de leer los mensajes y enviarlos al servidor de cola de mensajes en lapsos de tiempo regulares. El productor lee el registro de alarma y lo manda a un buffer, el consumidor lee del buffer cada cierto periodo de tiempo y manda el mensaje al servidor de cola de mensajes simulando la acción del gestor de fallas. El diagrama de bloques del simulador se puede apreciar en la figura 4.7.

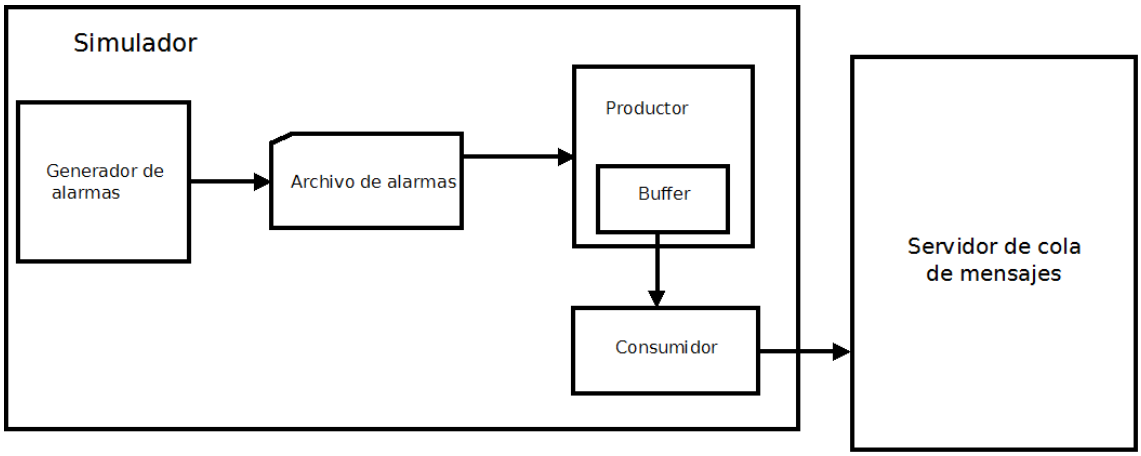

Figura 4.7: Diagrama de bloques del simulador de carga.

4.5 Monitor

Aunado al proceso de inferencia, es necesario tener un monitor que lleve el registro de las reglas activadas y el proceso que llevó a su activación. Para ello se elaboró un módulo monitor. Cada vez que una regla es activada y una acción ejecutada se enviará un mensaje a una cola de mensajes. Esta cola será leída por el monitor, el cuál presentará la información al usuario y almacenará la información en un archivo de texto. El diagrama de bloques del monitor se aprecia en la figura 4.8.

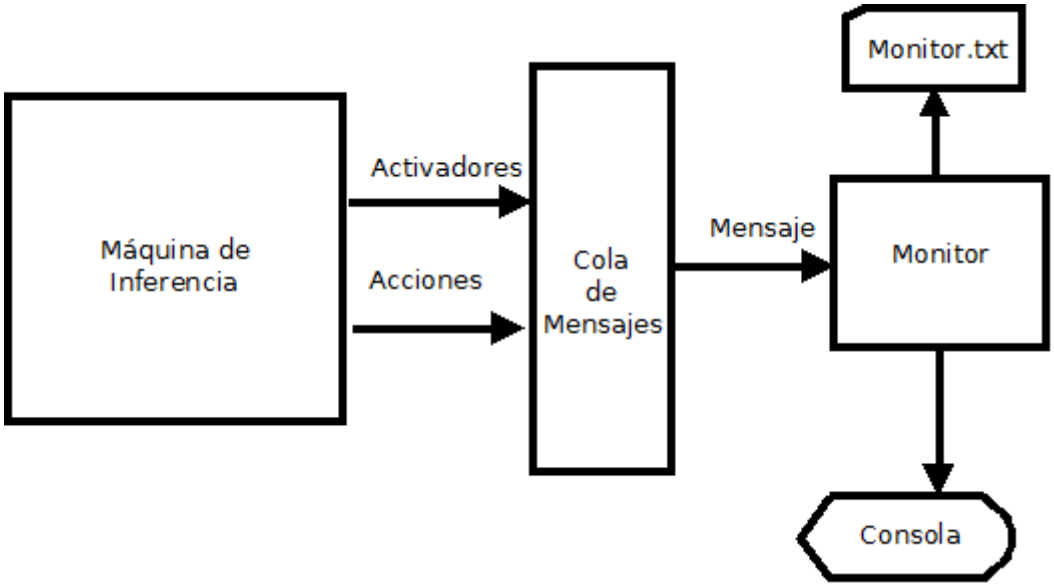

Figura 4.8 Diagrama de bloques del monitor de la máquina de inferencia.

5 Resultados de pruebas

Se procede a probar todos los tipos de regla, con el fin de verificar el correcto funcionamiento y cumplimiento con los requerimientos presentados anteriormente. Las pruebas son relativamente simples, teniendo en cuenta que la única manera de medir el desempeño real del sistema es en un ambiente de producción. Este tipo de evaluación esta fuera del alcance de este trabajo.

Además se considera que la interfaz de usuario y el editor de reglas son proyectos que si bien necesarios, no son indispensables para la evaluación del funcionamiento de la máquina de inferencia, por lo que se elaboró un pequeño módulo para la visualización de resultados, considerando que la elaboración de estos componentes forma parte del proceso de implementación del sistema en un ambiente de producción.

A continuación se presentan las pruebas realizadas a la máquina de inferencia desarrollada, estás se realizaron en un sistema operativo Windows Vista Service pack 1. Procesador Amd Athlon X2 y 3072 MB de memoria RAM DDR2 a 800MHz.

5.1 Base de reglas

Se elaboró una base de reglas de prueba, tomando en cuenta que se deben probar los tipos de eventos, las condiciones y las acciones mencionados en la sección 3.2.4. Para cualquier evento proveniente del gestor de fallas llamado "EVENTOGF", con los campos presentados en la tabla 3.1 tenemos.

Regla 1:

ACTIVADOR:
EVENTOGF.Texto IGUAL A "Falla en el sistema1"

EVALUADOR
Sin condición

ACCION
Mandar nuevo evento a gestor de fallas
EVENTONUEVO.Texto ="Falla en router"
Desactivar alarma.

Regla2:
ACTIVADOR:
EVENTOGF.Texto IGUAL A "Falla en el sistema1"

EVALUADOR
Sin condición

ACCION
Mandar mensaje “Desviar trafico” al servidor de acciones.

Regla3:
ACTIVADOR:
EVENTOGF.Texto IGUAL A “Falla en el sistema1”
Y
EVENTOGF.Severidad == 5

EVALUADOR
Si durante 5 minutos después
EXISTE EVENTO
EVENTO.Texto IGUAL A “Falla en el sistema2”
Y
EVENTOGF.OBJETO == EVENTO.OBJETO

ACCION
Mandar mensaje “Falla de conectividad en subred”

Regla4:
ACTIVADOR
EVENTOGF.Texto IGUAL A “Falla general1”

EVALUADOR
Si 5 minutos antes
EXISTE EVENTO
EVENTO.Texto IGUAL A “Previo falla general1”
AND EVENTOGF.OBJETO == EVENTO.OBJETO

ACCION
Mandar mensaje “Error de comunicación general”

Regla5:
ACTIVADOR
EVENTOGF.Texto IGUAL A “Error de conexion”

EVALUADOR
Si 5 minutos después
EXISTE EVENTO
EVENTO.Texto EQUALS “Error de conexion”
Y
CONEXIÓN(EVENTOGF.OBJETO,EVENTO.OBJETO)

ACCION
Mandar nuevo evento a memoria de trabajo
EVENTONUEVO.Texto EQUALS “Error de conectividad en red”

Regla 6:
ACTIVADOR
EVENTOGF.Texto IGUAL A “Eror de conectividad en red”

EVALUADOR
SI en Bitácora
EXISTE EVENTO
EVENTO.TEXTO IGUAL A “Falla en router”

ACCION
Mandar mensaje “Falla de conectividad en la red por caída de router”
Mandar nuevo evento al manejador de fallas
EVENTONUEVO.Texto EQUALS “Error de conectividad en red por router EVENTO.OBJETO”
EVENTONUEVO.SolucionProuesta EQUALS “Reemplazar router EVENTO.OBJETO”

Con esta base de regla se logra el objetivo de evaluar todas las funcionalidades de la máquina de inferencia como se explica a continuación:

- En la Regla 1 se prueba la ejecución de una acción al reconocer un evento. En este caso generamos un nuevo.

- En la Regla 2 se envía un mensaje al servidor de acciones al reconocer un evento.

- En la Regla 3 se detecta un evento, se inicia un periodo de espera de 5 minutos en busca de un evento correlacionado

- En la Regla 4 se realiza una correlación entre el evento detectado y un evento anterior que se encuentre en la memoria de trabajo.

- En la Regla 5 se realiza una correlación entre un evento y otro, además se analiza si el otro evento generado se origina en un objeto conectado al objeto originador del evento activador.

- En la Regla 6 se realiza un encadenamiento hacia adelante con la regla 5, si la acción de la regla 5 es ejecutada entonces la regla 6 será activada, esta regla buscará en la bitácora un evento correlacionado y procederá a enviar un evento al gestor de fallas.

Por lo tanto este pequeño conjunto de reglas contiene la funcionalidad total de la máquina de inferencia. A continuación se presentará el código de cada regla junto con los pasos que deberá seguir el generador de código en cada caso.

Regla 1:

```
    public class Regla1:IRule
    {
        public bool Activador(RegistroAlarma evento)
        {
            try
            {

                if (evento.Texto.Equals("Falla en el sistema1"))
                {
                    return true;
                }
                else
                    return false;
            }
            catch (Exception e)
            {
                return false;
            }
        }

        public bool Evaluador(RegistroAlarma evento, MemoriaDeTrabajo
memoria,string bitacora,string topologia)
        {
            /***Creación de evento****/
            RegistroAlarma respuesta = new RegistroAlarma();


            /*****Definición de evento*****/

            respuesta.Texto = "Falla en router";
            respuesta.Objeto = evento.Objeto;

            /********Accion**************/
            Acciones.EnviarEvento(respuesta);


            return true;

        }
    }
```

Aquí se aprecia una regla que recibe un evento y tan sólo ejecuta una acción, la acción consiste en enviar un evento al gestor de fallas, los paso requeridos son:

- Creación de una instancia de un registro de alarma
- Definición del evento, declarando los valores de los campos que se deseen.
- Enviar el evento al servidor de acciones, a través de la función EnviarEvento.

Regla 2:

```
    class Regla2:IRule
    {
        public bool Activador(RegistroAlarma evento)
        {
            try
            {

                if (evento.Texto.Equals("Falla en el sistema1"))
                {
                    return true;
                }
                else
                    return false;
            }
            catch (Exception e)
            {
                return false;
            }
        }

        public bool Evaluador(RegistroAlarma evento, MemoriaDeTrabajo
memoria,string bitacora,string topologia)
        {
            /***Creación de mensaje****/
            string mensaje = "Desviar Trafico falla en el objeto: ";
            mensaje += evento.Objeto;

            /********Accion**************/
            Acciones.EnviarMensaje(mensaje);


            return true;

        }
    }
```

Aquí al igual con la Regla 1, tan sólo se detecta un evento, y en este caso se envía un mensaje al servidor de acciones, por lo tanto los pasos son:

- Definir el contenido del mensaje
- Enviar el mensaje al servidor de acciones mediante la función EnviarMensaje.

Regla 3:

```
public class Regla3:IRule
    {
        DateTime tiempoInicial;
        DateTime tiempoActual;
        Boolean condicion1 = false;
        int umbral;

        public Regla3()
        {
            umbral = 5;

        }
```

```
        public bool Activador(RegistroAlarma evento)
        {
            try
            {

                if (evento.Texto.Equals("Falla en el sistema1") &&
evento.Severidad==5)
                {
                    return true;
                }
                else
                    return false;
            }
            catch (Exception e)
            {
                return false;
            }
        }

        public bool Evaluador(RegistroAlarma evento, MemoriaDeTrabajo
memoria,string bitacora,string topologia)
        {

            tiempoInicial = evento.Hora;
            tiempoActual = evento.Hora;

            System.Timers.Timer timer=new System.Timers.Timer(1000);
            timer.Elapsed+=new System.Timers.ElapsedEventHandler(timer_Elapsed);
            timer.Enabled=true;
            timer.Start();


            while ((tiempoActual - tiempoInicial) < TimeSpan.FromMinutes(umbral))
            {

                int busqueda =
                (from c in memoria
                 where c.Texto.Equals("Falla en el sistema2")
                     && c.Hora > tiempoInicial
                 select c).Count();

                if (busqueda>0)
                {
                    condicion1 = true;
                    break;
                }

            }


            /**********Evaluación de condiciones***************/
            if(condicion1)
            {

                /***Creación de mensaje****/
                string mensaje = "Falla de conectividad en subred";
                mensaje += evento.Objeto;

                /********Accion**************/
                Acciones.EnviarMensaje(mensaje);

            }
```

```
            timer.Dispose();
            return true;

        }


        private  void timer_Elapsed(object source, System.Timers.ElapsedEventArgs
e)
        {
            tiempoActual += TimeSpan.FromMilliseconds(1000);
        }
}
```

Aquí se correlaciona una alarma con un evento posterior en un lapso de cinco minutos, los pasos requeridos son:

- Declarar las variables:
  - tiempoInicial: Almacena la hora del evento detectado
  - tiempoActual: El tiempo transcurrido desde la detección del evento.
  - condicion1: Por cada condición a evaluar se crea una bandera, dicha bandera adquirirá el valor de verdadero si la condición se ha cumplido.
  - umbral: El umbral de tiempo determinado por el usuario, en este caso el umbral corresponde a 5 minutos.
- Se inicializa la variable umbral en el constructor por default de la regla.
- Se inicializan las variables tiempoInicial y tiempoActual en el cuerpo del método Evaluador.
- Se crea un temporizador que actualice el valor de la variable tiempoActual.
- Mientras no se cumpla el umbral de tiempo se realiza una búsqueda en la memoria de trabajo por un evento cuyo texto contenga la frase “falla en sistema2”. Con el fin de mejorar el rendimiento de la búsqueda se empleo la herramienta LINQ proporcionada por la plataforma .NET, que nos permite realizar la búsqueda en forma de una sentencia parecida a SQL. Cabe mencionar que la búsqueda puede realizarse a base de ciclos for y secuencias if, pero en este caso se aprovechan las facilidades ofrecidas por la plataforma.
- Si la búsqueda es exitosa el ciclo termina, incluso antes de cumplirse el umbral de tiempo, esto para permitir la atención inmediata al evento.
- Se evalúa si las condiciones han sido cumplidas, en cuyo caso se procede a realizar la acción, que corresponde a enviar un mensaje al servidor de acciones como se explicó en la Regla 2.

Regla 4:

```
    public class Regla4:IRule
    {
        int umbral;

        DateTime tiempoInicial;
        DateTime tiempoFinal;

        Boolean condicion1 = false;

        public Regla4()
```

```
        {
            umbral = 5;
        }

        public bool Activador(RegistroAlarma evento)
        {
            try
            {

                if (evento.Texto.Equals("Falla general1"))
                {
                    return true;
                }
                else
                    return false;
            }
            catch (Exception e)
            {
                return false;
            }
        }

        public bool Evaluador(RegistroAlarma evento, MemoriaDeTrabajo memoria,
string bitacora, string topologia)
        {


            tiempoFinal = evento.Hora;
            tiempoInicial = evento.Hora - TimeSpan.FromMinutes(umbral);

            int busqueda =
                (from c in memoria
                 where c.Texto.Equals("Previo Falla general1")
                     && evento.Objeto.Equals(c.Objeto)
                     && c.Hora > tiempoInicial && c.Hora < tiempoFinal
                 select c).Count();
            if (busqueda > 0)
                condicion1 = true;

            /**************Acciones******************/
            if (condicion1)
            {

                /***Creación de mensaje****/
                string mensaje = "Error de comunicacion general ";
                mensaje += evento.Objeto;

                /********Accion**************/
                Acciones.EnviarMensaje(mensaje);

            }
            return true;
        }
    }
```

Aquí se tiene una correlación entre un evento detectado y un evento anterior en la memoria de trabajo en un lapso de tiempo:

- Se declaran las variables
    - tiempoInicial correspondiente a 5 minutos antes de recibido un evento.
    - tiempoFinal la hora a la que fue recibido el evento.

- umbral el lapso de tiempo antes de que fuera recibido el evento.
- condicion1 bandera que indica que una condición ha sido cumplida.

- Se inicializa la variable umbral en el constructor por default
- En el método Evaluador se inicializa la variable tiempoFinal con la hora del evento, y la variable tiempoInicial 5 minutos antes.
- Se realiza la búsqueda por el evento cuyo texto es “Previo falla general1” y que además haya ocurrido 5 minutos antes del evento detectado. Otra vez se utiliza una secuencia LINQ para hacer más eficiente la búsqueda.
- Se verifican las condiciones cumplidas y se ejecuta la acción de enviar un mensaje al servidor de acciones.

Regla 5:

```
    public class Regla5:IRule
    {
        DateTime tiempoInicial;
        DateTime tiempoActual;
        int umbral;


        Boolean condicion1 = false;
        Boolean condicion2 = false;

        public Regla5()
        {
            umbral = 5;
            eventoActual = new RegistroAlarma();
        }
        public bool Activador(RegistroAlarma evento)
        {
            try
            {

                if (evento.Texto.Equals("Error de conexion"))
                {
                    return true;
                }
                else
                    return false;
            }
            catch (Exception e)
            {
                return false;
            }
        }

        public bool Evaluador(RegistroAlarma evento, MemoriaDeTrabajo
memoria,string bitacora,string topologia)
        {
            tiempoInicial = evento.Hora;
            tiempoActual = evento.Hora;


            System.Timers.Timer timer = new System.Timers.Timer(1000);
            timer.Elapsed += new
System.Timers.ElapsedEventHandler(timer_Elapsed);
            timer.Enabled = true;
            timer.Start();
```

```
            while ((tiempoActual - tiempoInicial) < TimeSpan.FromMinutes(umbral))
            {

                var busqueda =
                (from c in memoria
                 where c.Texto.Equals("No hay respuesta")

                     && c.Hora > tiempoInicial
                 select c.Objeto);
                if (busqueda.Count() > 0)
                    condicion1 = true;

                foreach (string objeto in busqueda)
                {
                    if (OperacionesTopologia.Conexion(topologia, evento.Objeto,
objeto))
                    {
                        condicion2 = true;
                        break;
                    }
                }
                if (condicion1 && condicion2)
                    break;

            }


            /**********Evaluación de condiciones***************/
            if (condicion1&& condicion2)
            {

                /***Creación de evento****/
                RegistroAlarma respuesta = new RegistroAlarma();


                /*****Definición de evento*****/

                respuesta.Texto = "Error de conectividad en red";
                respuesta.Objeto = evento.Objeto;
                respuesta.Hora = evento.Hora;
                /********Accion**************/
                Acciones.EnviarAMemoria(respuesta,memoria);



            }
            return true;

        }


        private void timer_Elapsed(object source, System.Timers.ElapsedEventArgs
e)
        {
            tiempoActual += TimeSpan.FromMilliseconds(1000);
        }

    }
```

Aquí se observa una correlación entre un evento y otro posterior, además se analiza si los objetos emisores de los eventos están conectados. El proceso es análogo a la

Regla3, con una diferencia, se realizan dos búsquedas dentro del ciclo while, la primera búsqueda sólo se preocupa por encontrar un evento en la memoria de trabajo, una vez que se encuentran todos los eventos con el texto correspondiente, se procede a realizar una segunda sobre los resultados de la primera búsqueda. Debido a que esta segunda búsqueda hace uso de la función "Conexion" para probar la conexión entre 2 objetos, no es posible utilizar sentencias LINQ, por lo que la búsqueda se realiza a base de un ciclo for y sentencias if. Si las 2 búsquedas entregan resultados se ejecuta la acción de enviar un evento a la memoria de trabajo, a través de la función "EnviarAMemoria".

Regla 6:

```
    public class Regla6:IRule
    {
        DateTime tiempoInicial;
        DateTime tiempoFinal;
        DateTime tiempoActual;
        int umbral;

        Boolean condicion1;

        public Regla6()
        {
            umbral = 5;
        }

        public bool Activador(RegistroAlarma evento)
        {
            try
            {

                if (evento.Texto.Equals("Error de conectividad en red"))
                {
                    return true;
                }
                else
                    return false;
            }
            catch (Exception e)
            {
                return false;
            }
        }

        public bool Evaluador(RegistroAlarma evento, MemoriaDeTrabajo
memoria,string bitacora,string topologia)
        {
            /***************Definicion de variables****************/
            tiempoFinal = evento.Hora;
            tiempoInicial = tiempoFinal - TimeSpan.FromMinutes(umbral);

            /*********Conexion a bitacora*************/
            String conection = bitacora;
            SqlConnection conn = new SqlConnection(conection);


            /************Creacion de query************/
            string query = "Select * from bitacora where Texto='Falla en router'
and Hora between @tiempoInicial and @tiempoFinal";
```

```
            /************Ejecución de query****************/
            SqlCommand comm = new SqlCommand(query, conn);
            comm.Parameters.Add(new
SqlParameter("@tiempoInicial",tiempoInicial));
            comm.Parameters.Add(new SqlParameter("@tiempoFinal",tiempoFinal));
            conn.Open();
            SqlDataReader reader = comm.ExecuteReader();


            if (reader.HasRows)
            {
                condicion1 = true;
            }
            conn.Close();

            /**********Evaluación de condiciones**************/
            if (condicion1)
            {
                /***Creación de evento****/
                RegistroAlarma respuesta = new RegistroAlarma();


                /*****Definición de evento*****/

                respuesta.Texto = "Falla de conectividad en red por
router"+evento.Objeto;
                respuesta.Objeto = evento.Objeto;
                respuesta.SolucionPropuesta = "Reemplazar router " +
evento.Objeto;

                /********Accion**************/
                Acciones.EnviarEvento(respuesta);

                /***Creación de mensaje****/
                string mensaje = "Falla de conectividad en la red por caida de
router";
                mensaje += evento.Objeto;

                /********Accion**************/
                Acciones.EnviarMensaje(mensaje);

            }


            return true;

        }

    }
```

En la regla 6, la condición de activación depende del resultado de la acción de la regla 5, si esta regla emite un registro de alarma conformado por el texto “Error de conectividad en la red” el emparejador de eventos lo detectará y mandará a llamar al Evaluador de la regla. Ahí se realiza una consulta a la bitácora en busca de un registro en un lapso de tiempo de 5 minutos antes de la emisión del mensaje de alerta. Además de las variables de tiempoInicial y tiempoFinal utilizadas en reglas anteriores es necesario abrir una conexión a la base de datos para realizar consultas sobre la

bitácora. Luego se realiza una consulta buscando registros con el texto “Falla en Router” ocurridos entre el tiempoInicial y el tiempoFinal utilizando la clausula BETWEEN de SQL.

Si la búsqueda retorna resultados se da la condición como cumplida y se procede a ejecutar acciones de manera análoga a las demás reglas.

5.3 Pruebas

Como modelo de topología se creó un ejemplo que consta de 154 nodos y 367 enlaces. Para la creación del ejemplo, se utilizó como base el modelo presentado en (31), el cual fue desarrollado para la herramienta de simulación de tráfico de red “TOTEM” (26). Fue necesario adaptar la topología al formato reconocido por la máquina de inferencia. Dentro de dicha topología se identificaron dos objetos conectados, específicamente los objetos 79 y 152. Aquí se presentan ambos objetos y su conexión en la topología.

```
<Topology>
        <nodes>
                .
                .
                .
                <object id="79">
                </object>
                .
                .
                .
                </object>
                <object id="152">
                .
                .
                .
        </nodes>
        <links>
                .
                .
                .
                <link id="79_9 -> 152_0">
                        <from id="79" />
                        <to id="152" />

                </link>
                .
                .
                </links>
</topology>
```

Se procedió a realizar un caso de prueba consistente en 7 registros ideados para activar y ejecutar todas las reglas del caso de prueba, utilizando los objetos 79 y 152 para probar la evaluación de condición de conexión entre objetos:

%a
-TipoDeSystema=TIPOG
-TipoDeRegistro=13
-IdRegistro=0
-Objeto=O35
-Severidad=5
-Hora=6/1/2009 4:37:00 PM
-TipoDeEvento=9
-Causa=1
-ProblemaEspecifico=8
-Texto=Previo Falla general1
-NumeroDeAlarma=1204849616
-ClaseDeAlarma=14
-Status=0
-SolucionPropuesta=solucion6
%a
%a
-TipoDeSystema=TIPOG
-TipoDeRegistro=13
-IdRegistro=1
-Objeto=O35
-Severidad=5
-Hora=6/1/2009 4:37:00 PM
-TipoDeEvento=9
-Causa=1
-ProblemaEspecifico=8
-Texto=Falla en el sistema1
-NumeroDeAlarma=1204849616
-ClaseDeAlarma=14
-Status=0
-SolucionPropuesta=solucion6
%a
%a
-TipoDeSystema=TIPOG
-TipoDeRegistro=13
-IdRegistro=2
-Objeto=O35
-Severidad=5
-Hora=6/1/2009 4:37:00 PM
-TipoDeEvento=9
-Causa=1
-ProblemaEspecifico=8
-Texto=Falla en el sistema2
-NumeroDeAlarma=1204849616
-ClaseDeAlarma=14

-Status=0
-SolucionPropuesta=solucion6
%a
%a
-TipoDeSystema=TIPOG
-TipoDeRegistro=13
-IdRegistro=3
-Objeto=O35
-Severidad=5
-Hora=6/1/2009 4:38:00 PM
-TipoDeEvento=9
-Causa=1
-ProblemaEspecifico=8
-Texto=Falla general1
-NumeroDeAlarma=1204849616
-ClaseDeAlarma=14
-Status=0
-SolucionPropuesta=solucion6
%a
%a
-TipoDeSystema=TIPOG
-TipoDeRegistro=13
-IdRegistro=4
-Objeto=152
-Severidad=5
-Hora=6/1/2009 4:38:00 PM
-TipoDeEvento=9
-Causa=1
-ProblemaEspecifico=8
-Texto=Falla en router
-NumeroDeAlarma=1204849616
-ClaseDeAlarma=14
-Status=0
-SolucionPropuesta=solucion6
%a
%a
-TipoDeSystema=TIPOG
-TipoDeRegistro=13
-IdRegistro=5
-Objeto=152
-Severidad=5
-Hora=6/1/2009 4:38:00 PM
-TipoDeEvento=9
-Causa=1
-ProblemaEspecifico=8
-Texto=Error de conexion
-NumeroDeAlarma=1204849616
-ClaseDeAlarma=14
-Status=0
-SolucionPropuesta=solucion6

%a
%a
-TipoDeSystema=TIPOG
-TipoDeRegistro=13
-IdRegistro=6
-Objeto=79
-Severidad=5
-Hora=6/1/2009 4:38:00 PM
-TipoDeEvento=9
-Causa=1
-ProblemaEspecifico=8
-Texto=No hay respuesta
-NumeroDeAlarma=1204849616
-ClaseDeAlarma=14
-Status=0
-SolucionPropuesta=solucion6
%a

Una vez guardados en un archivo los casos de prueba, se procedió a ejecutar el simulador el cual envía los mensajes al servidor de cola de mensajes, para verificar el correcto funcionamiento del servidor, se consulta la interfaz administrativa del mismo. En la figura 5.1 se observa la el contenido de la cola "test_load4" dicha cola almacena los mensajes provenientes del administrador de fallas (simulador en este caso). Los 7 mensajes correspondientes a los 7 registros de alarmas ideados se encuentran almacenados en la cola.

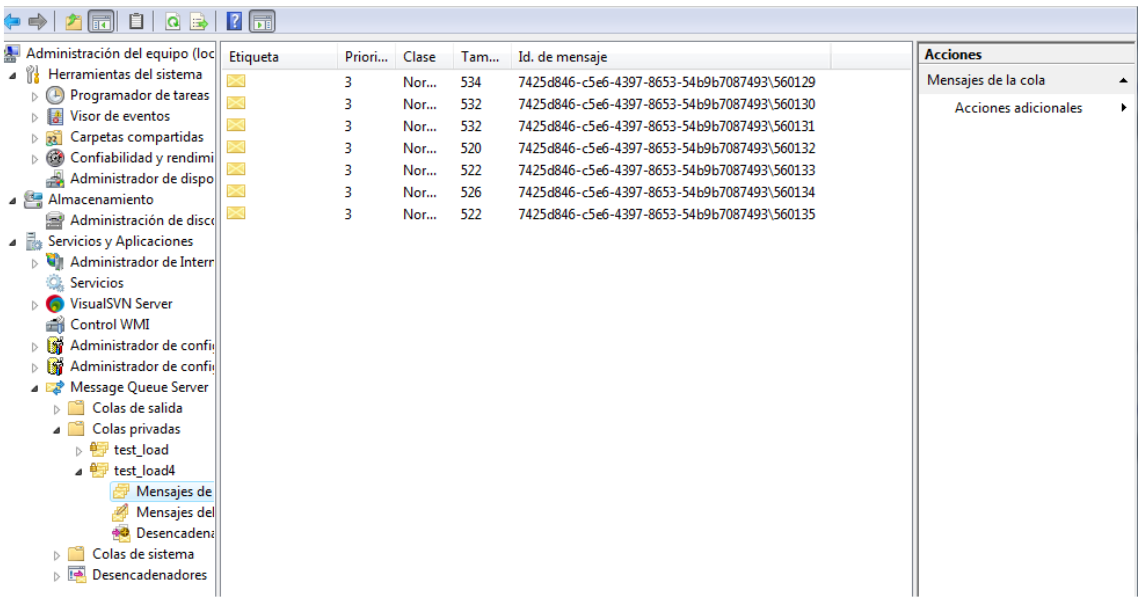

Figura 5.1: Estado de la cola de mensajes después de ejecutar el simulador.

Se procedió a eliminar los mensajes de la cola, ya que en un ambiente de monitoreo en tiempo real, el sistema tiene que estar operando de manera paralela a la recepción de mensajes, por lo que la prueba se debe realizar con la interfaz de control de mensajes y la máquina de inferencia ejecutándose.

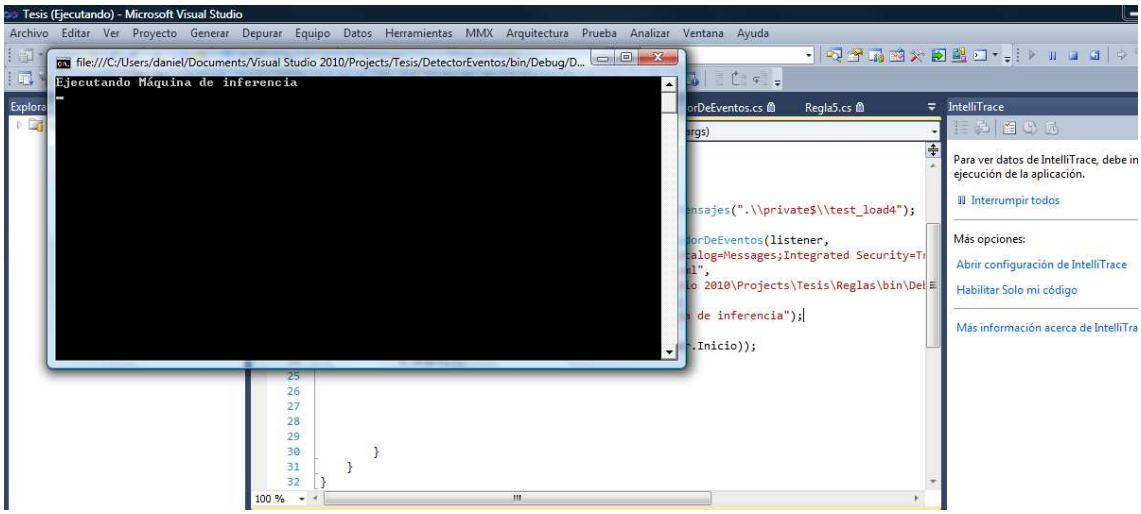

Figura 5.2: Ambiente de desarrollo y pantalla de ejecución de la máquina de inferencia.

Posteriormente se ejecutó el simulador, el cual, envía mensajes a la cola a una tasa de 120 mensajes por minuto (2 mensajes por segundo), que es la tasa de operación esperada en este tipo de sistemas.

La figura 5.3 presenta el estado de la tabla "Bitácora" de la base de datos de alertas, como se puede apreciar, la tabla presenta 7 registros, correspondientes a los 7 registros enviados por el simulador.

IdRegis	TipoDeSystema	TipoDeRegis	Objeto	Severidad	Hora	TipoDeEvento	Causa	Problema	Texto
0		13	O35	5	06/01/2009 04:37:00 p.m.	9	1	8	Previo Falla general
1		13	O35	5	06/01/2009 04:37:00 p.m.	9	1	8	Falla en el sistema1
2		13	O35	5	06/01/2009 04:37:00 p.m.	9	1	8	Falla en el sistema2
3		13	O35	5	06/01/2009 04:38:00 p.m.	9	1	8	Falla general1
4		13	152	5	06/01/2009 04:38:00 p.m.	9	1	8	Falla en router
5		13	152	5	06/01/2009 04:38:00 p.m.	9	1	8	Error de conexion
6		13	79	5	06/01/2009 04:38:00 p.m.	9	1	8	No hay respuesta

Figura 5.3: Tabla Bitácora después de ejecutado el simulador.

La figura 5.4 presenta la pantalla correspondiente a la salida del programa Monitor de la máquina de inferencia. Aquí se encuentra la información de el momento en que una regla es activada por un evento, así como la hora y evento Activador, análogamente, para la ejecución de acciones también es añadido un registro en el archivo. Al analizarlo se observa la hora y evento Activador para las reglas 1,2,3,4,5 y 6. Así como las acciones tomadas por cada regla. Finalmente la figura 5.5 presenta el contenido de la cola de mensajes "test_load", la cual corresponde a la comunicación con el servidor de acciones. En la figura se aprecian 6 mensajes, correspondientes a las acciones ejecutadas por las reglas (mensajes enviados al servidor de acciones).

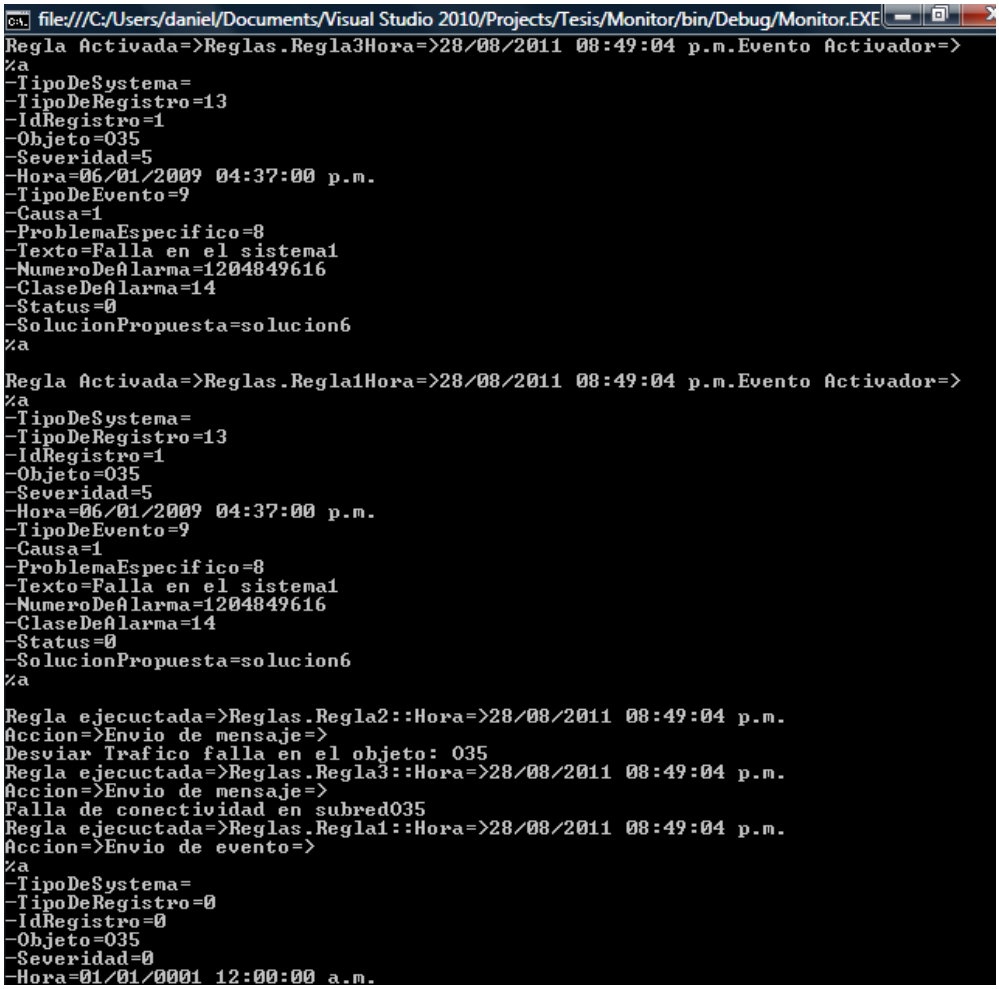

Figura 5.4: Mensajes del programa Monitor de la máquina de inferencia.

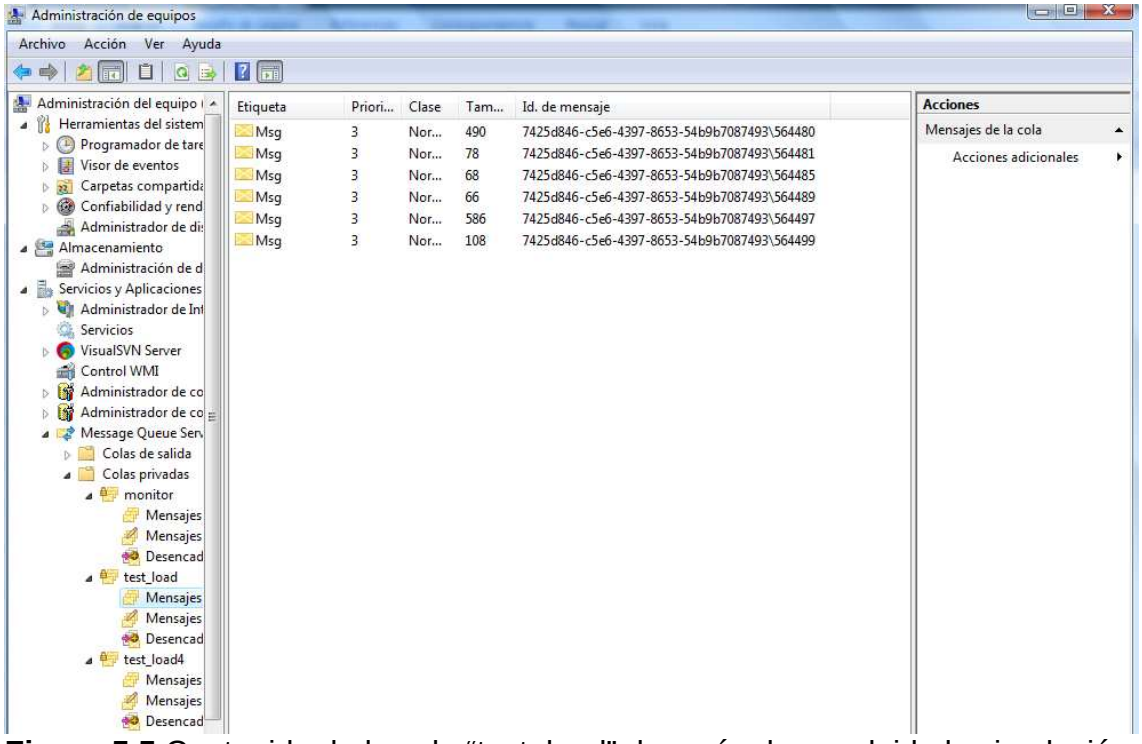

Figura 5.5:Contenido de la cola “test_load” después de concluida la simulación.

Los resultados muestran que el proceso de inferencia se llevó a cabo conforme a lo esperado para el caso de prueba.

Ahora bien se generó un archivo de prueba con mil registros con valores aleatorios que representan 5 minutos de operación del sistema gestor de fallas. Acto seguido se sustituyeron 7 de estos registros de alarmas por el caso de prueba, procurando que el campo hora de estos registros concuerde con los precedentes y antecedentes.

Con este nuevo archivo de pruebas se ejecutó el simulador, los resultados son análogos al caso de prueba, ya que al no poder emparejar evento alguno, no se activaron las reglas. En cuanto el sistema recibió el caso de prueba el proceso de inferencia se realizó de manera idéntica al experimento con nada más 6 registros. Sin embargo sí se detectó un pequeño retraso en la atención de los mensajes. Esto gracias a la cola de "test_load4", en operación normal la cola se encuentra vacía o con un mensaje, ya que los mensajes están siendo atendidos instantáneamente. Pero en esta ocasión, cuando el monitor comenzó a registrar un proceso de activación de reglas, en la cola se hicieron presentes mensajes sin atender como se aprecia en la figura 5.6, dicho retraso continuó hasta concluida la labor del simulador. La tabla 5.1 muestra un pequeño reporte de dicha simulación elaborado a partir de la bitácora del monitor.

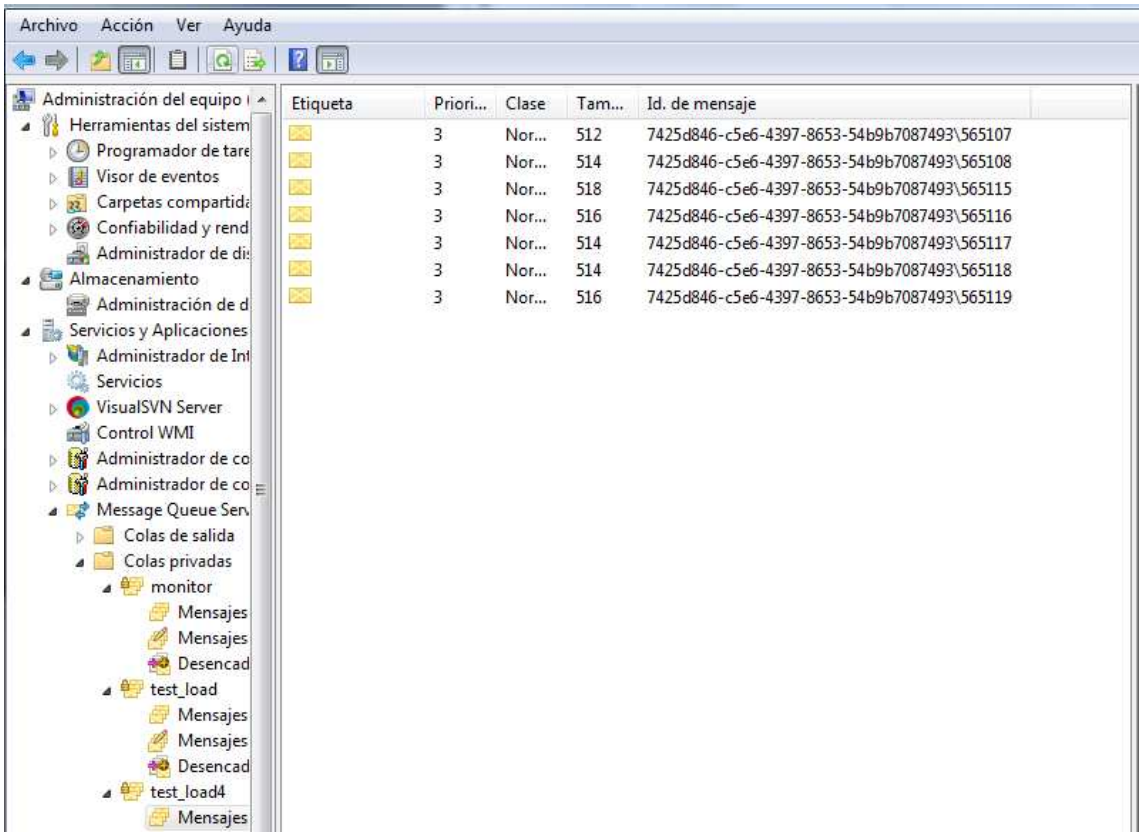

Figura 5.6: Contenido de la cola “test_load4” donde se aprecia un retraso de 7 mensajes por atender por la máquina de inferencia.

Regla Activada	Hora de activación	Id evento activador	Hora de ejecución	Acción ejecutada
Regla 1	6:51:17 p.m.	1011	06:51:17 p.m.	Envío de evento
Regla 2	6:51:17 p.m.	1011	06:51:17 p.m.	Envío de mensaje
Regla 3	6:51:17 p.m.	1011	06:51:18 p.m.	Envío de mensaje
Regla 4	06:51:18 p.m.	1013	06:51:18 p.m.	Envío de mensaje
Regla 5	06:51:19 p.m.	1015	06:51:20 p.m.	Envío de evento a memoria
Regla 6	06:51:20 p.m.	Regla 5	06:51:20 p.m.	Envío de mensaje Envío de evento

Tabla 5.1: Reporte de activación de alertas según los contenidos de la bitácora del monitor de eventos.

6 Conclusiones

Una red TMN consiste en una infraestructura de manejo y administración de una red de telecomunicaciones. Como parte de esta infraestructura el proceso de manejo y atención a fallas es esencial.

El manejo de fallas es un proceso complejo, que involucra la atención a miles de mensajes provenientes de diferentes equipos, dichos equipos no siempre pertenecen al mismo fabricante e incluso entre equipos de un mismo fabricante existen formatos de mensajes heterogéneos.

En la actualidad existen soluciones comerciales para manejo y atención de fallas, estos sistemas son llamados "Administradores de Fallas" o "Gestores de Fallas". Los administradores de fallas se pueden considerar como unificadores, que reciben mensajes de diferentes fuentes, a través de un conjunto de interfaces de comunicación e intérpretes. Una vez recibidos por el Administrador los mensajes son traducidos a un formato estándar y se realizan operaciones básicas de atención y filtrado.

Aún así, el número de mensajes generados por un gestor de fallas se encuentra en el orden de miles de mensajes por hora. Un operador humano difícilmente podrá atender todos los mensajes y descubrir patrones e información relevante en el conjunto de mensajes.

Con el fin de mejorar el desempeño del proceso de gestión de fallas se plantea el desarrollo de un sistema experto para sistemas de telecomunicaciones. Dicho sistema se encargará de la tarea de correlacionar, filtrar y atender las alarmas pertinentes, dependiendo de un conjunto de reglas basadas en la experiencia de los operadores humanos. Más que un reemplazo a un operador humano, será un herramienta para asistir en la tarea de gestión de fallas. Actualmente existe un pequeño número de soluciones comerciales de este tipo, más están limitadas a un producto o familia de productos de telecomunicaciones.

El elemento fundamental de dicho sistema, y tema de estudio del presente trabajo es la máquina de inferencia. La máquina de inferencia lleva el control del proceso de razonamiento integrando los eventos de alarma recibidos para que puedan ser analizados por las reglas, a su vez, se encarga de regular la activación de reglas. Las reglas son expresiones de la forma "si-entonces" que representan el conocimiento del experto en el ramo.

La máquina de inferencia actúa bajo el ciclo Emparejamiento-Selección-Acción. En la etapa de emparejamiento se contrastan las condiciones de una regla con los hechos conocidos. En el caso de la máquina desarrollada se optó por un esquema de reglas orientadas a eventos. Bajo este esquema se añade una condición al cuerpo de la regla, dicha condición se cumple cuando se detecta un evento, dicho evento se llamará "Activador".

La máquina desarrollada realiza un emparejamiento y una evaluación, primero se contrasta el evento recibido del gestor de fallas contra los eventos activadores de cada regla. Este emparejamiento se realiza en paralelo, es decir el mismo evento se está contrastando con el evento activador en cada regla en un proceso o hilo diferente. Si el resultado del emparejamiento regresa un valor lógico de verdadero, se procede a activar la regla, para la segunda fase de evaluación de condiciones.

En la segunda fase se toman en cuenta condiciones de naturaleza heterogénea, teniendo como fuentes de información: una representación de la topología de la red, una estructura de datos que contiene eventos recientes y eventos que el experto considere que deben mantenerse en memoria, y una base de datos que contiene un historial de todas las alertas recibidas por el gestor de fallas. A través del método de búsqueda secuencial se realiza una búsqueda de las condiciones satisfechas. Si todas las condiciones son cumplidas se ejecuta la acción de la regla de manera asíncrona, es decir cada regla ejecuta su acción, que consiste en enviar un mensaje a un servidor, sin esperar a que las demás reglas concluyan con su proceso de evaluación.

La máquina de inferencia tiene integrado un mecanismo de encadenamiento hacia adelante, es decir la acción de una regla puede ser la generación de un evento que activa otra regla de manera directa (a través de la memoria de trabajo) o indirecta (por medio del sistema gestor de fallas).

Añadido a la máquina de inferencia se desarrolló la infraestructura necesaria para que funcione de acuerdo a las necesidades detectadas. Esta infraestructura incluye la estructura de datos para almacenar los eventos activos, llamada memoria de trabajo. La memoria de trabajo es una lista sincronizada, al momento de ejecutar las acciones de insertar y eliminar elementos, el proceso o hilo que ejecuta la acción adquiere un "seguro" sobre la lista, impidiendo que otros procesos (reglas en este caso) modifiquen la memoria de trabajo hasta que la acción esté concluida. Además se le dotó de mecanismos de notificación que informarán al emparejador de eventos cuando se añaden o eliminan elementos, e incluso proporciona la facilidad a las reglas para recibir dichas notificaciones de ser necesario.

También se desarrolló una base de datos que contendrá la bitácora de eventos recibidos por la máquina de inferencia. Esta base de datos es un análogo a una memoria de largo plazo, contendrá todas las alarmas recibidas en una tabla, la cual puede ser analizada por las reglas a través de una secuencia select en lenguaje SQL.

Finalmente se diseñó una representación de la topología de la red, la topología es un modelo simplificado de la red con sus elementos (en forma de objetos) y sus relaciones de contención, herencia y conexión. Se eligió un formato xml para la representación de la topología, con dos grupos de elementos, "nodos" que representan los elementos de la red, y "links" que representan las conexiones entre los nodos de la red. La topología será accesible para las reglas a través de una interfaz xml.

A su vez se desarrollo un simple simulador, su fin es emular los mensajes provenientes del sistema gestor de fallas. El simulador consta de dos etapas, un generador de

mensajes, que crea un conjunto de mensajes de manera aleatoria y los guarda en un archivo de texto. La segunda etapa lee el archivo de texto y mediante una arquitectura productor consumidor, manda mensajes a la máquina de inferencia en intervalos de tiempo regulares. En caso de ser necesario se puede ejecutar una simulación a partir de un archivo editado manualmente, sin la intervención del generador.

Una vez teniendo todos los elementos disponibles se realizó una prueba del sistema a partir de una pequeña base de reglas. En el sistema experto las reglas se codificarán a través de un editor de reglas, el editor presentará un ambiente de trabajo para que el experto cree reglas de operación las cuales se almacenarán en un archivo xml a partir del cual un intérprete generará código. Dicho código será compilado en una biblioteca dinámica. Esta biblioteca es en efecto la base de reglas, cada regla estando representada por una clase con sus propios atributos y métodos, siendo obligatoria únicamente la implementación de una interfaz. El editor de reglas e intérprete de código no fue desarrollado, pero se presentaron las bases para la creación de código en diferentes casos.

La máquina de inferencia lee la base de reglas a través de una técnica conocida como reflexión, crea una instancia cada regla y ejecuta los miembros de la interfaz que son Activador y Evaluador, que corresponden a la primera y segunda etapa del proceso de inferencia respectivamente.

La base de reglas de prueba consiste en 6 reglas, las cuales ponen a prueba cada una de las funcionalidades de la máquina de inferencia, se presentó el código de cada una así como notas sobre el posible método de generación de código.

Se creó un caso de ejemplo para probar el correcto funcionamiento de la máquina de inferencia. Se generó un archivo con mensajes de alerta diseñados para activar cada una de las reglas.

Se verificó el correcto funcionamiento de la máquina de inferencia, sobre todo en los casos más críticos, en los cuales una vez detectado un evento activador se inicia un período de tiempo de espera en busca de un evento específico. Se comprobó que en este caso el proceso de inferencia continúa, por lo que se evitó este posible cuello de botella gracias al proceso de evaluación asíncrono de reglas.

Se observó un retraso de aproximadamente 7 mensajes en la atención a eventos con cargas de trabajo constantes a una taza de 120 mensajes por minuto. Cabe destacar que el Hardware empleado para realizar las pruebas dista mucho del empleado en un ambiente de producción, por lo que pruebas en dicho ambiente son necesarias para determinar el desempeño del sistema en un ambiente de telecomunicaciones real.

Por lo tanto se concluye que para cumplir con las exigencias de tiempo real para el problema planteado, una arquitectura con emparejamiento en paralelo y ejecución asíncrona puede desempeñarse de de manera adecuada siempre y cuando se tengan bien definidos los mecanismos de sincronización.

También se observó que es necesario construir un conjunto de herramientas de apoyo y fuentes de información para la máquina de inferencia. Sin éstos elementos complementarios, la máquina no puede realizar operaciones de correlación complejas en el tiempo (lapso entre dos eventos) y el espacio (eventos conectados o relacionados en la red). Dichas herramientas están conformadas por un conjunto de tecnologías diversas, como son bases de datos, lenguajes de maquetado, bibliotecas dinámicas, generadores de código y comunicación entre procesos.

6.1 Trabajo futuro

El alcance de este trabajo consistió en generar un prototipo viable y bien diseñado con perspectiva a su implementación en una red de telecomunicaciones real, con la respectiva exigencia de rendimiento y funcionalidad.

Este trabajo presenta el diseño y prototipo de la máquina de inferencia, más es necesario realizar pruebas en un ambiente real de telecomunicaciones y con reglas elaboradas por expertos para hacer las adaptaciones necesarias.

Adicionalmente el sistema experto en telecomunicaciones requiere la elaboración de un editor de reglas y generador de código. Dicha tarea en sí, puede ser motivo de otro proyecto, ya que se requiere una interfaz amigable e intuitiva para el experto, pero sin sacrificar expresividad para aprovechar las facilidades proporcionadas por la máquina de inferencia.

También es necesario realizar un análisis profundo del consumo de recursos del sistema en el ambiente de producción. El sistema desarrollado no pone límite al número de hilos que pueden ser iniciados por la máquina de inferencia, sin embargo existe la posibilidad de que un número elevado de hilos influya de manera negativa en el rendimiento del sistema. Por lo que sería necesario realizar adaptaciones a la máquina instaurando un límite de hilos en ejecución.

Una vez que el sistema esté listo para su implementación, se requiere la elaboración de módulos específicos al ambiente. En especial adaptaciones a los formatos de datos y elaboración de un servidor de acciones que reciba los mensajes y eventos de la máquina de inferencia y los traduzca a acciones sobre el sistema gestor de fallas y/o notificaciones al operador de la red TMN.

La elaboración del sistema experto para manejo de fallas en sistemas de telecomunicaciones es un camino arduo y complejo. En este trabajo se presentó una base para solución a este problema.

Bibliografía

1. **Aiko, Ron Sprenkels.** *Introduction to TMN.* University of Twente. 1999.

2. **Ng, Gee Wah.** *Intelligent Systems Fusion, Tracking and Control.* s.l. : RESEARCH STUDIES PRESS LTD.

3. **Hopgood, Adrian A.** *Intelligent Systems for Engineers and Scientists.* s.l. : CRC Press, 2001.

4. **Marcellin Jacques Sergio**, *Notas del Curso: "Construcción de Sistemas Expertos"*, Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, 2010.

5. **J, G Giarratano,.** *Expert Systems: Principles and Programming.*

6. **Ioannis, Nick Bassiliades.** *Parallel object-oriented and active knowledge base systems.* s.l. : Kluwer Academic Publishers Boston/Dordecht/London, 1998.

7. **Batory, Don.** *The LEAPS Algorithm.* 1994, Technical Report. University of Texas at Austin, Austin, TX, USA.

8. **Forgy, Charles.** *Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem.* 1982, Artificial Intelligence, 19, pp. 17-37.

9. **Miranker, Daniel P.** *TREAT: A New and Efficient Match Algorithm for AI Production Systems.* 1990, Pitman/Morgan Kaufmann.

10. **Dibble, Peter C.** *Real-Time Java Platform Programming.* s.l. : Prentice Hall PTR, 2002.

11. **P., D. Emilio.** *Incorporación de un sistema basado en reglas en un entorno en tiempo real.* Universidad Politecnica de Valencia. 2004.

12. **David, Ashok K.Agrawala.** *The challenges of Real-Time AI.* U. Maryland ¿. 1994.

13.**Greg Stanley and associates** Rule-based approaches and implementation[citado Septiembre 20 2011]. *http://www.gregstanleyandassociates.com/whitepapers/FaultDiagnosis/Rules/rules.htm*.

14. **Ishida, Toru.** *Parallel Rule Firing in Production Systems.* 1991, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, Vol. 3, pp. 11-17.

15. **Neiman, Daniel.***Control Issues in Parallel Rule-Firing Production Systems.* 1991. pp. 310-316.

16. **Gruschke, Boris.** *INTEGRATED EVENT MANAGEMENT: EVENT CORRELATION USING DEPENDENCY GRAPHS.* 1998.

17. **IBM ILOG.** [Citado: Agosto 25, 2011.] Pagina oficial ILOG. http://www-01.ibm.com/software/websphere/ilog/.

18. **HP TemIP**. [Citado: Agosto 25, 2011.] Descripción de HP TeMIP expert.. http://sysdoc.doors.ch/HP/5981-4399EN.pdf.

19. **G2 Platform Real-Time Business Rules Engine (BRE) for Mission-Critical Applications**. [Citado:Enero 13 2011] Pagina g2 gensym. http://149.75.206.5/index.php?option=com_content&view=article&id=51&Itemid=58.

20. **Wikipedia.** CLIPS. [Online] 08 25, 2011. Pagina en wikipedia sobre CLIPS. http://en.wikipedia.org/wiki/CLIPS.

21. **Production System Technologies**. [Citado: Agosto 25, 2011.] Pagina de Production System Technologies.. http://www.pst.com/rete2.htm.

22. **Jess**. [Citado: Agosto 25, 2011.] http://www.jessrules.com/. http://www.jessrules.com/.

23. **Drools expert**. [Citado: Agosto 25, 2011.] http://downloads.jboss.com/drools/docs/5.0.1.26597.FINAL/drools-expert/html_single/index.html.

24. **Implementacion de LEAPS por Drools**. [Citado: Agosto 25, 2011.] http://www.jbug.jp/trans/jboss-rules3.0.2/ja/html/ch01s05.html. http://www.jbug.jp/trans/jboss-rules3.0.2/ja/html/ch01s05.html.

25. **G2 Success Stories**. [citado Enero 13 2011]. http://149.75.206.5/index.php?option=com_content&view=article&id=245:ericsson&catid=83:network-management&Itemid=152.

26. **TOTEM project (TOolbox for Traffic Engineering Methods)**. [Cited: Septiembre 20, 2011.] http://totem.info.ucl.ac.be/.

27. **Fault Manager Expert** *Fault Manager Basic Functions, Developer's Guide.* s.l. : Sony Ericsson.

28. **Malenfant, M. Jaques J.** *A tutorial on behavioral reflection and it's implementation.*

29. **Smith, Brian Cantwell.** *Procedural Reflection in programming languages.* s.l. : Department of Electrical Engineering and Computer Sciences, Massachussetts Institute of Technology, 1982. phd Thesis.

30. **Microsoft MSDN.** [Citado: Septiembre 20,2011.] http://msdn.microsoft.com/es-es/library/system.eventhandler(v=vs.80).aspx.

31. **Heckmann, Oliver.** [Citad: Agosto 27, 2011.] http://dmz02.kom.e-technik.tu-darmstadt.de/~heckmann/.