# Capítulo 6

# Sistema de validación

Debido a que las funcionalidades del sistema activo de tolerancia a fallas deben ser probadas y evaluadas, son necesarios la definición e implementación de un sistema de validación que permita probar y evaluar dichas funcionalidades.

Este capítulo presenta un sistema de validación para el sistema activo de tolerancia a fallas definido en este trabajo. Consiste de un sistema distribuido de transporte que simula un sistema de ensamble de vehículos.

El capítulo está organizado de la siguiente manera. La sección 6.1 describe la arquitectura y el funcionamiento del sistema distribuido de transporte. La sección 6.2 explica la forma en que el sistema activo de tolerancia a fallas y el sistema distribuido de transporte se integran. La sección 6.3 expone diferentes pruebas realizadas en los sistemas. La sección 6.4 concluye el capítulo destacando las ventajas y dificultades encontradas en el uso del sistema activo de tolerancia a fallas con el sistema de validación.

## 6.1 Contexto de experimentación

### 6.1.1 Sistema distribuido de transporte

El sistema de validación para el servicio de tolerancia a fallas es un sistema distribuido de transporte. Dicho sistema implementa la simulación de una fábrica encargada de ensamblar automóviles a partir de las diferentes piezas producidas por diferentes proveedores.

La fábrica y los proveedores se ubican en un mundo virtual organizado en cuadrantes. Las piezas necesarias para la construcción de un auto son transportadas por entidades móviles que se desplazan en el mundo. Dada una solicitud de un cliente para la compra de un auto, el objetivo es recolectar todas las piezas requeridas que lo conforman contactando a los proveedores que proporcionan cada una de las piezas.

### 6.1.2 Arquitectura

La figura 6.1 muestra la arquitectura del sistema distribuido de transporte.

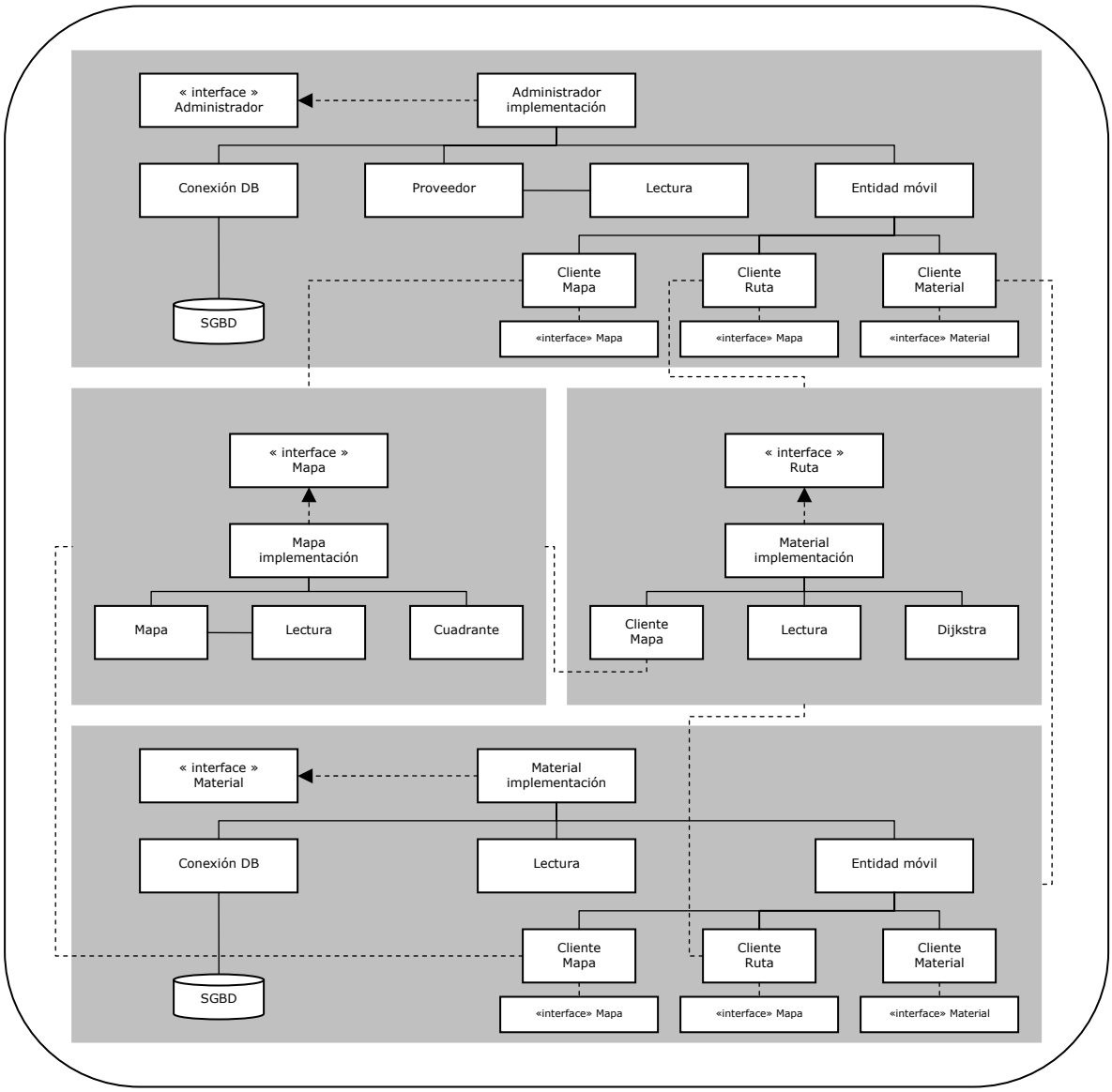



*Figura 6.1 Arquitectura del sistema distribuido de transporte*

El sistema distribuido de transporte fue implementado en java con ayuda de RMI para la invocación remota de métodos. Su arquitectura se basa principalmente en el modelo definido por RMI.

La arquitectura presenta cuatro módulos principales: (i)el servidor administrador, (ii) el servidor de mapa, (iii) el servidor de ruta y (iv) el servidor de material. Es importante aclarar que cada módulo está situado en una localidad diferente, no comparte los mismos recursos, ni accede a las mismas bases de datos que los otros módulos.

### 6.1.3 Módulos de operación

La figura 6.2 muestra los componentes del sistema distribuido de transporte.

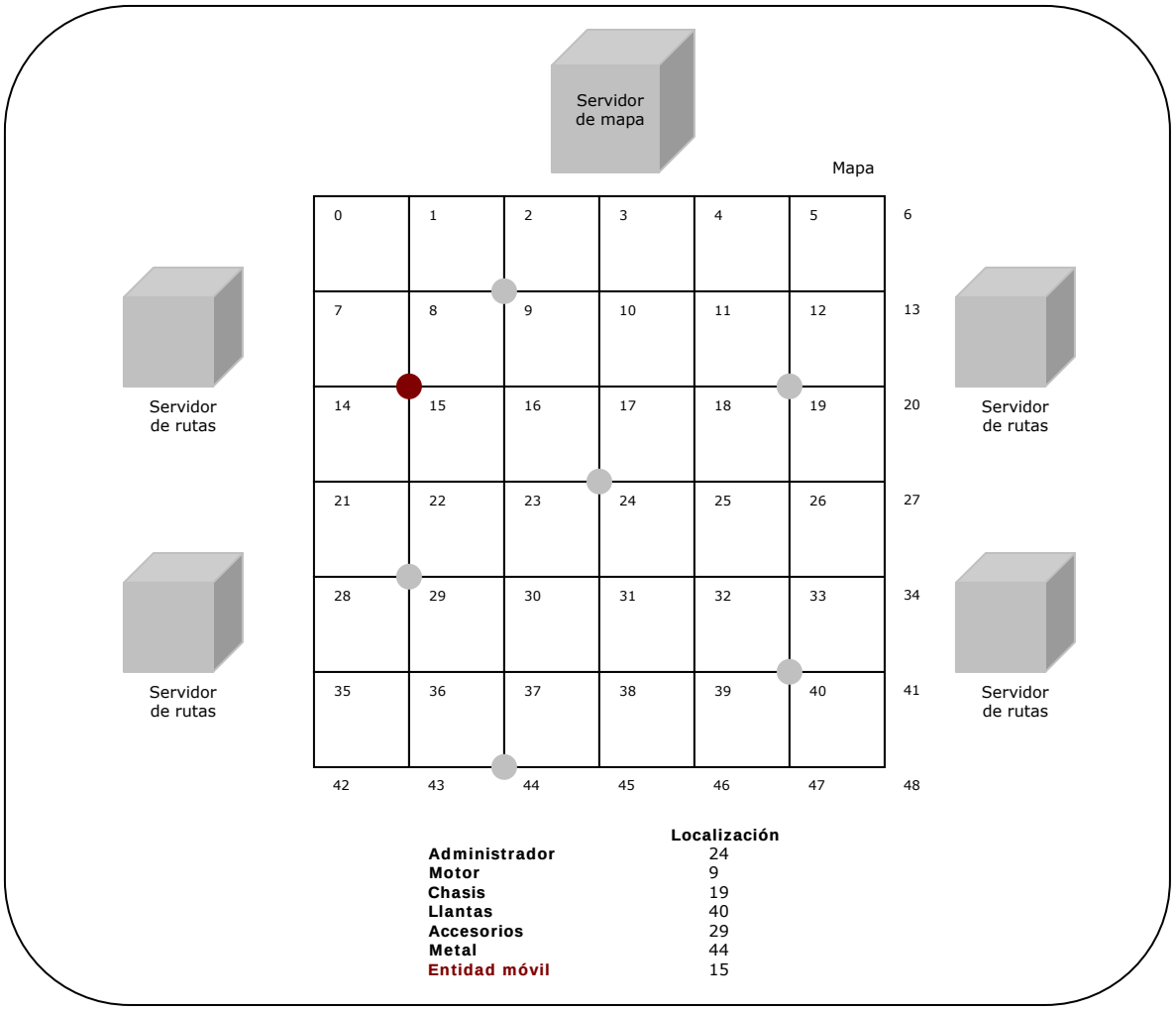



*Figura 6.2 Componentes del sistema distribuido de transporte*

A continuación se enlistan los componentes del sistema.

- Mapa: es la representación de una localidad física (interior de una fábrica) donde diferentes objetos pueden tener una ubicación dada en coordenadas. Para llegar de un punto del mapa a otro es necesario recorrer un camino. El mapa está dividido en cuadrantes, donde cada uno de ellos puede tener una topología diferente.

- Servidores

    - Los servidores de información proporcionan sus servicios a módulos u objetos pertenecientes al mundo virtual. No tienen una localización dentro del mapa.

        - El servidor administrador mantiene el catálogo con los tipos de automóviles que la fábrica ensambla y las piezas que los conforman.

Recibe las solicitudes por parte de los clientes y se encarga del proceso de ensamblado a partir de la generación de entidades móviles que recolectan las piezas requeridas para el ensamble.

- El servidor de mapa provee una descripción de la topología de cada cuadrante así como información de los módulos que se localizan dentro de cada uno.

- Para cada cuadrante del mapa existe un servidor de rutas. Por lo tanto, cada servidor de rutas contiene la descripción de la topología de un cuadrante y calcula una ruta óptima entre dos puntos que pertenecen a éste.

- Los servidores de material tienen una localización dentro del mapa, proveen las piezas que conforman a un automóvil y sólo pueden proporcionar su servicio si la entidad móvil que realiza la petición se encuentra en su misma ubicación.

  En base al contexto aplicativo, cinco servidores de material han sido definidos para el sistema distribuido de transporte: (i) servidor de accesorios, (ii) servidor de chasis, (iii) servidor de llantas, (iv) servidor de metal y (v) servidor de motor.

- Entidades móviles: objetos que se ubican dentro del mapa, creados por ciertos servidores para la recolección de piezas, por lo tanto, tienen la capacidad de desplazarse de una coordenada a otra siguiendo caminos (ruta crítica) y transportar cargas (materiales).

El funcionamiento del sistema se describe como la realización de una petición vía Web de un automóvil específico por parte de un cliente al servidor administrador, el cual se encarga de iniciar el proceso de ensamble del automóvil.

La figura 6.3 muestra el diagrama de secuencias del proceso de ensamble de un automóvil.

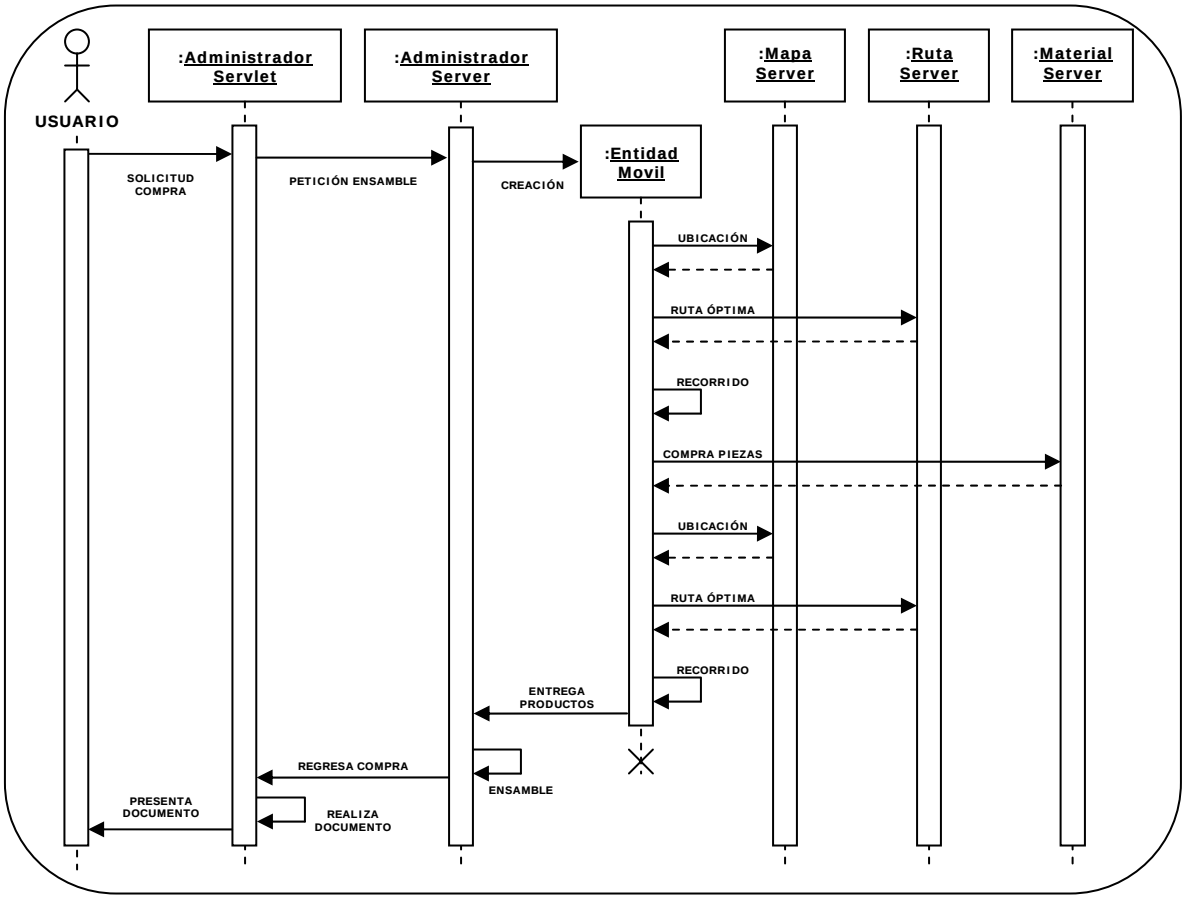



*Figura 6.3 Diagrama de secuencias del ensamble de un automóvil*

De manera general, dada la solicitud de compra de un automóvil, el proceso de ensamble está conformado por los siguientes pasos:

1. El servidor administrador crea el conjunto de entidades móviles necesarias para recuperar las piezas que conforman el automóvil. Cada entidad móvil creada recibe una lista de piezas y el nombre del proveedor correspondiente que las produce (servidor de material).

2. Las entidades móviles se comunican con el servidor de mapa para conocer la ubicación de los servidores de material que tienen que visitar.

3. Las entidades móviles se comunican con el servidor de ruta del cuadrante donde se encuentran localizadas. Este servidor les proporciona la ruta óptima entre su ubicación y la ubicación del servidor de material especificado, para poder llevar a cabo un recorrido. Si es necesario, las entidades móviles se comunicarán con diferentes servidores de ruta conforme vayan cambiando de cuadrante.

4. Cuando una entidad móvil llega a la ubicación de un servidor de material realiza la petición del número de piezas que su lista especifica. El servidor de material proporciona la cantidad exacta del material solicitado. Si dicho servidor no cuenta

con el material suficiente y requiere de los servicios de otro servidor de material para ensamblar el material requerido, crea entidades móviles que recolecten las piezas que necesita para proporcionar su servicio.

5. Las entidades móviles que tienen su lista completa regresan al servidor administrador auxiliándose de los servidores de ruta como en el paso 3. Las entidades móviles que aún no tengan todo lo que su lista les indica deben de repetir los pasos 3 y 4.

El proceso de recolección termina cuando todas las entidades móviles creadas han recolectado sus productos y los han entregado al servidor administrador.

## 6.2 Implementación

Debido a la necesidad de comunicación remota entre los módulos o servidores del sistema distribuido de transporte, su implementación se realizó en base a la tecnología RMI de Java.

### 6.2.1 Servidor administrador

El servidor administrador responde a las peticiones remotas del administrador Web a través de la realización de las siguientes funciones implementadas: **obtención de la lista de proveedores** registrados, **adición de un automóvil** para un proveedor, **eliminación de un automóvil**, **obtención de la lista de los automóviles** registrados, la **descripción** y las **lista de las piezas** que conforman un automóvil.

La clase ***AdministradorInterfaz*** presenta la interfaz del servidor administrador del sistema distribuido de transporte que declara los métodos de comunicación remota con los otros servidores del sistema.

La clase ***AdministradorImpl*** implementa los métodos declarados en la interfaz del servidor para ofrecer sus servicios. Esta clase también proporciona el resto del código que configura el servidor: un método *main* que crea e instala un controlador de seguridad, crea un objeto remoto y lo registra.

La clase ***AdministradorLectura*** implementa un mecanismo de lectura para los archivos públicos de definición de *host* y número de puerto donde el servidor administrador es iniciado.

La clase ***AdministradorConectaDB*** implementa los métodos de conexión a la base de datos del servidor administrador para realizar altas, bajas y consultas.

El apéndice B presenta los diagramas de éstas clases implementadas por el sistema distribuido de transporte.

El servidor administrador **recibe** la **petición** de compra de un automóvil y con respecto a la cantidad de piezas que lo conforman y a la capacidad de sus entidades móviles, **crea** las

**entidades móviles** necesarias con una lista de piezas a ser recolectadas para el ensamble del automóvil.

La clase ***EntidadMovil***, descrita en el apéndice B, implementa el objeto entidad móvil creado por el servidor administrador.

Las entidades móviles **solicitan** al servidor de mapa la **localización** de los proveedores que producen las piezas solicitadas y a los servidores de ruta la **ruta óptima** para llegar a dicha localización, **realizan el recorrido** y **compran las piezas**. Cuando una entidad móvil ha recolectado las piezas solicitadas regresa a la localización del servidor administrador, para lo cual solicita a los servidores de ruta la ruta óptima y realizan el recorrido final para **entregar las piezas**.

Debido a que las entidades móviles se comunican de manera remota con otros servidores que conforman el sistema, es necesario establecer clientes a dichos servidores y que éstos proporcionen sus interfaces para establecer comunicación remota entre ambas partes.

Las clases ***AccesoriosCliente***, ***ChasisCliente***, ***LlantasCliente***, ***MetalCliente***, ***MotorCliente***, ***MapaCliente***, ***RutaCuadrante1Cliente***, ***RutaCuadrante2Cliente***, ***RutaCuadrante3Cliente*** y ***RutaCuadrante4Cliente*** implementan los clientes de los servidores correspondientes en el servidor administrador. El apéndice B presenta los diagramas de éstas clases implementadas por el sistema distribuido de transporte.

Las clases ***AccesoriosInterfaz***, ***ChasisInterfaz***, ***LlantasInterfaz***, ***MetalInterfaz***, ***MotorInterfaz***, ***MapaInterfaz***, ***RutaCuadrante1Interfaz***, ***RutaCuadrante2 Interfaz***, ***RutaCuadrante3Interfaz*** y ***RutaCuadrante4Interfaz*** presentan las interfaces de los servidores que declaran los métodos de comunicación remota con una entidad móvil del servidor administrador. El apéndice B presenta los diagramas de éstas clases implementadas por el sistema distribuido de transporte.

Las clases ***AccesoriosLectura***, ***ChasisLectura***, ***LlantasLectura***, ***MetalLectura***, ***MotorLectura***, ***MapaLectura***, ***RutaCuadrante1Lectura***, ***RutaCuadrante2 Lectura***, ***RutaCuadrante3Lectura*** y ***RutaCuadrante4Lectura*** implementan la lectura de los archivos públicos de definición de *host* y número de puerto de los servidores. El apéndice B presenta los diagramas de éstas clases implementadas por el sistema distribuido de transporte.

El servidor administrador **recibe** la **solicitud** de las piezas requeridas a una entidad móvil que ha recolectado todas las piezas en su lista y la **elimina**. Una vez que se han recolectado todas las piezas necesarias se realiza el **ensamble** del automóvil y la petición de compra de un auto se da por finalizada exitosamente.

### 6.2.2 Servidor de mapa

Un grafo G(V,E) consiste de un conjunto V de elementos (vértices) y un conjunto E de pares desordenados de miembros de V (*edges* o aristas), donde los vértices de un grafo son los puntos y las aristas son las líneas que conectan cada par de puntos. Por lo tanto, el mapa del sistema distribuido de transporte puede ser tratado como un grafo para calcular la ruta óptima entre dos puntos.

Un mapa es creado en base a la definición del número de cuadrantes, del número de vértices, de los caminos en los cuadrantes (aristas entre pares de vértices y su distancia), de los vértices frontera entre cuadrantes y de la localización de los servidores físicos. La clase ***CreacionMapa*** implementa el proceso de creación del mapa creando los objetos cuadrante correspondientes. La clase ***Cuadrante*** implementa el objeto cuadrante conformado por el nombre del cuadrante, los caminos que definen su topología y los servidores que se localizan en éste. El apéndice B presenta los diagramas de éstas clases implementadas por el sistema distribuido de transporte.

La figura 6.4 presenta el mapa definido para el sistema distribuido de transporte.

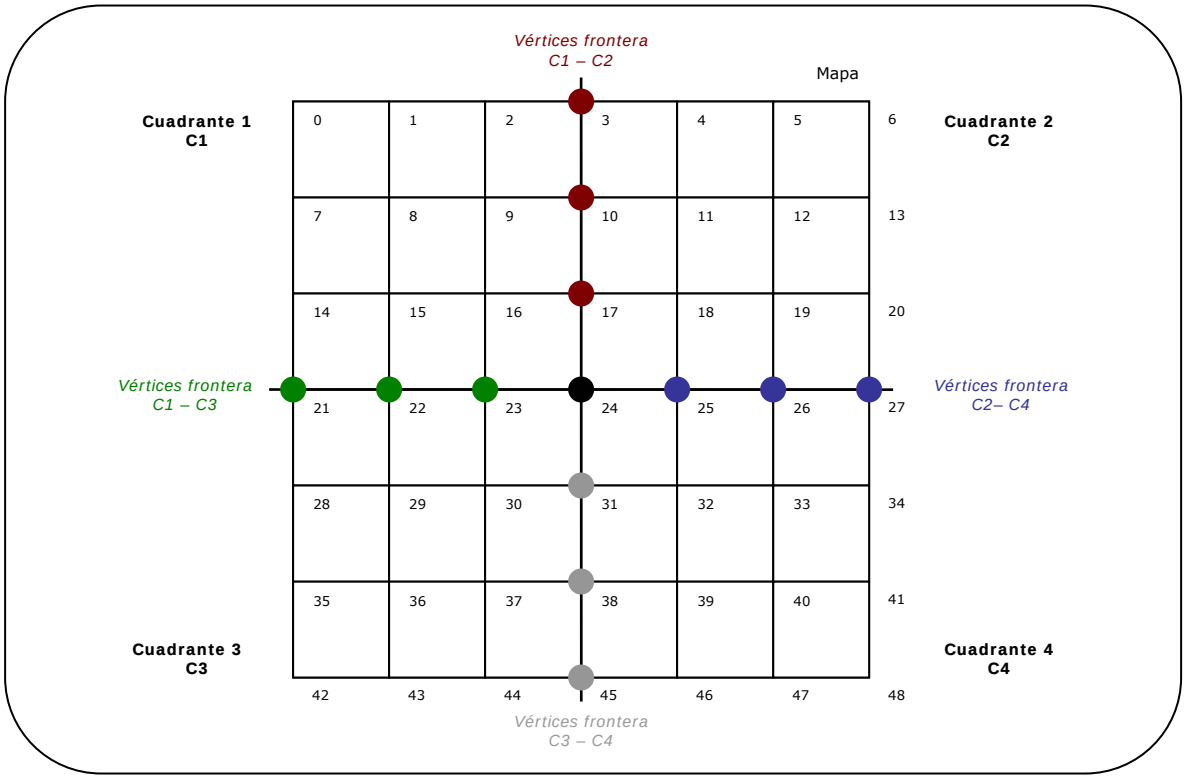



*Figura 6.4 Mapa del sistema distribuido de transporte.*

La definición para la creación del mapa del sistema distribuido de transporte comprende 48 vértices. El mapa está dividido en cuatro cuadrantes con caminos específicos y conectados con los otros cuadrantes a través de los vértices frontera.

La clase ***MapaInterfaz*** presenta la interfaz del servidor de mapa del sistema distribuido de transporte que declara los métodos de comunicación remota con otros servidores del sistema.

La clase ***MapaImpl*** implementa los métodos declarados en la interfaz del servidor; de igual forma, proporciona el resto del código que lo configura.

El servidor de mapa proporciona, a través de la invocación remota de sus métodos implementados, el cuadrante al cual pertenece un proveedor a las entidades móviles creadas por servidores y los vértices frontera entre dos cuadrantes a los servidores de ruta.

La clase ***MapaLectura*** implementa un mecanismo de lectura para los archivos públicos de definición de *host* y número de puerto donde el servidor de mapa es iniciado.

El apéndice B presenta los diagramas de éstas clases implementadas por el sistema distribuido de transporte.

### 6.2.3 Servidor de ruta

De acuerdo a la topología definida para el mapa, cuatro servidores de ruta (uno para cada cuadrante) han sido implementados.

Las clases ***RutaCuadrante1Interfaz***, ***RutaCuadrante2Interfaz***, ***RutaCuadrante 3Interfaz*** y ***RutaCuadrante4Interfaz*** presentan las interfaces de los servidores de rutas del sistema distribuido de transporte correspondientes a cada cuadrante que declara los métodos de comunicación remota con los otros servidores del sistema. El apéndice B presenta los diagramas de éstas clases implementadas por el sistema distribuido de transporte.

Las clases ***RutaCuadrante1Impl***, ***RutaCuadrante2Impl***, ***RutaCuadrante3Impl*** y ***RutaCuadrante4Impl*** implementan los métodos declarados en su interfaz correspondiente para ofrecer sus servicios. También proporcionan el resto del código que configura el servidor. El apéndice B presenta los diagramas de éstas clases implementadas por el sistema distribuido de transporte.

Las clases ***RutaCuadrante1Lectura***, ***RutaCuadrante2Lectura***, ***RutaCuadrante3 Lectura*** y ***RutaCuadrante4Lectura*** implementan los mecanismos de lectura para los archivos públicos de definición de *host* y número de puerto donde los servidores correspondientes son iniciados. El apéndice B presenta los diagramas de éstas clases implementadas por el sistema distribuido de transporte.

El algoritmo *Dijkstra* es el algoritmo implementado para calcular la ruta óptima de un punto origen a un punto destino. La implementación de este algoritmo requiere considerar los siguientes aspectos: (i) identificar un vértice origen, (ii) identificar un vértice destino, (iii) etiquetar los vértices del grafo con un valor real (inicialmente infinito), (iv) manejar una cola de prioridad que es ordenada en base en el peso del camino más corto desde el vértice origen al vértice destino, y (v) cuando un elemento es insertado en la cola de prioridad y ya existe, la copia previa debe ser removida y la nueva debe ser insertada en el nivel correcto.

El seudo código del algoritmo *Dijkstra* es el siguiente:

```
S ← ∅
S' ← V
for all (i ∈ V) do
      d(i) ← INFINITO
end for
```

```
d(1) ← 0
pred(1) ← 0
while (|s| < m) do
    Sea i ∈ S' el nodo para el cual  d(i) =min{ d(j) :  j ∈ S'}
    S ← S ∪ { i }
    S' ← S' \ { i }
    for ( (i,j) ∈ (i)) do
      if ( (d,j) >  d(i) + c_ij ) then
            d(j) ← d(i) + c_ij
            pred(j) ← i
      end if
    end for
end while
```

Las clases ***Arista***, ***camino***, ***Cuadrante***, ***Dijkstra***, ***Grafo***, ***Nodo*** y ***Vertice*** descritas en el apéndice B, implementan el algoritmo para el cálculo de la ruta óptima entre dos puntos.

Por lo tanto, a través de la invocación remota de sus métodos que interactúan con el servidor de mapa, los servidores de ruta proporcionan a una entidad móvil la ruta óptima entre un punto origen y un punto destino.

### 6.2.4 Servidores de material

Los servidores de accesorios, chasis, llantas, metal y motor proporcionan a las entidades móviles los productos solicitados actualizando su base de datos. La implementación de las clases de estos servidores es similar en base a su funcionalidad.

Las clases ***<ServidorMaterial>Interfaz*** presentan las interfaces de los servidores de material del sistema distribuido de transporte que declaran los métodos de comunicación remota con los otros servidores del sistema.

Las clases ***<ServidorMaterial>Impl*** implementan los métodos declarados en la interfaz del servidor correspondiente para ofrecer sus servicios. También proporcionan el resto del código que configura el servidor.

Las clases ***<ServidorMaterial>ConectaDB*** implementa los métodos de conexión a la base de datos del servidor correspondiente para realizar consultas y actualizaciones.

Las clases ***<ServidorMaterial>Lectura*** implementan un mecanismo de lectura para los archivos públicos de definición de *host* y número de puerto donde los servidores son iniciados.

El apéndice B presenta los diagramas de éstas clases implementadas por el sistema distribuido de transporte.

Cuando un servidor de material no cuenta con la cantidad necesaria de piezas para satisfacer la solicitud de un cliente, es capaz de **crear entidades móviles** encargadas de recolectar las piezas necesarias para el **ensamble** de las piezas solicitadas. Cada servidor de material implementa en la clase ***EntidadMovil*** un objeto entidad móvil correspondiente a su contexto.

Por lo tanto, los servidores de material a través de sus entidades móviles creadas establecen comunicación remota con otros servidores del sistema, para lo cual implementa clientes a dichos servidores y requiere de las interfaces que declaran los métodos de éstos.

### 6.2.5 Compra electrónica

El proceso de compra electrónica de un automóvil para el sistema es implementado mediante el uso de servlets y JSPs (*JavaServer Pages*) y la herramienta Tomcat de Apache.

Los servlets son programas que se ejecutan en un servidor Web, que fungen como una capa intermedia entre una petición proveniente de un navegador Web u otro cliente http, y las bases de datos o aplicaciones del servidor HTTP. Su finalidad es la de leer los datos enviados por el usuario, buscar cualquier otra información respecto a la petición, generar los resultados, dar formato a los resultados dentro de un documento (insertar la información en una página HTML en la mayoría de los casos), establecer los parámetros HTTP de respuesta adecuados y devolver el documento al cliente.

Algunas razones por las que las páginas Web requieren ser generadas al momento son que la página Web está basada en los datos enviados por el usuario, deriva de datos que cambian constantemente y utiliza información de bases de datos u otros orígenes que se encuentran en el servidor.

La tecnología de las *JavaServer Pages* permite mezclar el HTML regular y estático con un contenido dinámicamente generado a partir de los servlets.

Tomcat es un contenedor de servlets con entrono JSP. Puede ser usado como un pequeño servidor autónomo para probar los servlets y las páginas JSP, o puede estar integrado dentro del servidor Web Apache.

Por lo tanto, el proceso de compra electrónica está compuesto por las siguientes actividades.

- Mostrar el catálogo de automóviles de un proveedor
- Mostrar la descripción de un automóvil elegido del catálogo
- Comprar un automóvil
- Registrar proveedores
- Agregar automóviles al catálogo de un proveedor
- Borrar automóviles del catálogo de un proveedor

La clase ***Administrador*** implementa el servlet que permite realizar las actividades de compra electrónica de un automóvil a través de la invocación remota de los métodos del servidor administrador.

### 6.2.6 Inicio del sistema

Para el inicio del sistema distribuido de transporte se ha creado un administrador del sistema que concentra las operaciones necesarias. Mediante el uso de la herramienta ANT

de Apache y archivos de configuración XML el administrador ejecuta las tareas necesarias para el inicio del sistema.

El administrador permite, a través de la GUI que proporciona, iniciar uno o más de los módulos que conforman al sistema con el objetivo de demostrar la distribución de los mismos. Esta interfaz provee al usuario un ambiente amigable gracias al uso de funciones de la librería Swing de Java.

Para iniciar el sistema distribuido de transporte es necesario establecer un TCP/IP (RMI Registry) por cada máquina virtual utilizada, que permite establecer la conectividad entre funciones remotas.

Las clases implementadas, descritas en el apéndice B, para la GUI del sistema son:

***AdministradorSDT:*** GUI que permite iniciar el sistema de manera rápida y sencilla a través del menú de opciones y botones que proporciona, y seguir su funcionamiento a través de la impresión en las terminales del sistema.

***Dialogo:*** presenta un contenedor de diálogo al usuario. Esta clase presenta dos personalizaciones del contenedor: (i) para la obtención del número de puerto TCP/IP en el cual el sistema será iniciado, y (ii) para presentar la información acerca del sistema.

***Escritura:*** crea un archivo que contiene el número de puerto del sistema activo de tolerancia a fallas para uso público.

***Terminal:*** componente que simula una terminal del sistema.

***Impresion:*** imprime información en las terminales del sistema.

***RMIRegistry:*** inicia el RMI Registry en un puerto establecido y lo finaliza a través del uso de la clase *NamingService*.

***NamingService:*** inicia el RMI Registry para un número de puerto establecido (comprobando que el puerto no esté siendo utilizado) y lo finaliza.

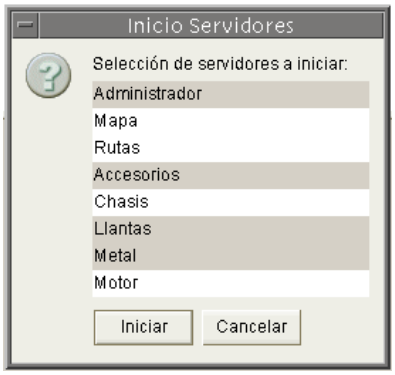
***NamingServiceMBean:*** interfaz implementada por la clase *NamingService*.

La figura 6.5 muestra la GUI que proporciona el administrador del sistema distribuido de transporte.


**Diálogo de obtención de número de puerto RMI**

*Figura 6.5 Interfaz gráfica del administrador del sistema distribuido de transporte*


**Menú del sistema**




**Opción de *Iniciar Servidores***

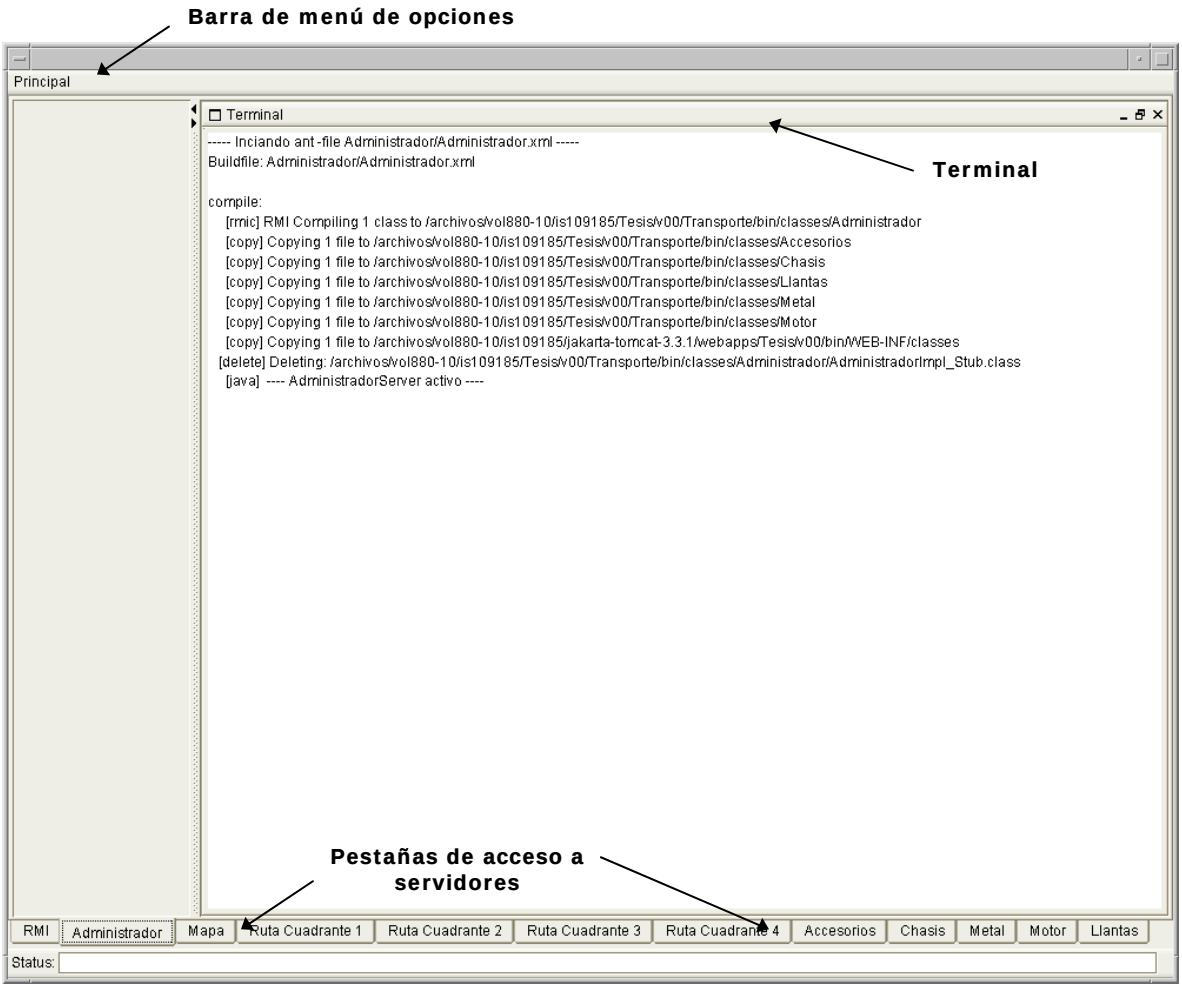



*Figura 6.5 Interfaz gráfica del administrador del sistema distribuido de transporte (continuación)*

La clase ***AdministradorSDT***, descrita en el apéndice B, implementa el administrador del sistema distribuido de transporte.

Para el inicio de la ejecución de dos o más servidores dentro de la aplicación que el administrador proporciona es necesario que sean iniciados en un mismo registro (RMIRegistry) debido a que son ejecutados en una máquina virtual de Java que no permite a una aplicación levantar más de un registro.

Una vez iniciado el sistema distribuido es necesario arrancar el servidor Tomcat para que los usuarios tengan acceso al sitio Web de venta de automóviles.

Tomcat utiliza de forma predeterminada un puerto que no es estándar pera evitar conflictos con los servidores Web que ya se encuentren en uso. El número de puerto utilizado actualmente es el 8085.

## 6.3 Integración

El sistema activo de tolerancia a fallas establece ciertos requerimientos para que una aplicación distribuida le sea integrada y sea tolerante a fallas.

El sistema activo proporciona su interfaz a las aplicaciones distribuidas para que puedan acceder a sus métodos de manera remota. La clase ***SistemaTFInterfaz*** presenta la interfaz que el sistema activo de tolerancia a fallas proporciona al sistema distribuido de transporte, que declara los métodos que permiten la comunicación remota entre el sistema distribuido y el sistema activo de tolerancia a fallas.

La clase ***SistemaTFCliente*** implementa en el sistema distribuido el cliente del sistema activo de tolerancia a fallas que accede de manera remota a los métodos implementados declarados en su interfaz. El apéndice B presenta los diagramas de éstas clases implementadas por el sistema distribuido de transporte.

El sistema distribuido de transporte, a través de comunicación remota, configura el sistema activo de tolerancia a fallas a través de la proporción de un archivo XML de configuración. Un ejemplo de la definición de un archivo XML por parte del sistema distribuido de transporte para la configuración del sistema activo de tolerancia es:

```
<configuracion>
  <nivelTF>3</nivelTF>
  <Entidad nombre="ADMINISTRADOR" tolerancia="Si" suscripcion="Si">
    <comunicacion>
    <entidades>
        <nombreEntidad>MAPA</nombreEntidad>
    </entidades>
    <entidades>
        <nombreEntidad>RUTACUADRANTE1</nombreEntidad>
    </entidades>
    <entidades>
        <nombreEntidad>RUTACUADRANTE2</nombreEntidad>
    </entidades>
    <entidades>
        <nombreEntidad>RUTACUADRANTE3</nombreEntidad>
    </entidades>
    <entidades>
        <nombreEntidad>RUTACUADRANTE4</nombreEntidad>
    </entidades>
```

```
    <entidades>
        <nombreEntidad>MOTOR</nombreEntidad>
    </entidades>
    <entidades>
        <nombreEntidad>CHASIS</nombreEntidad>
    </entidades>
    <entidades>
        <nombreEntidad>LLANTAS</nombreEntidad>
    </entidades>
    <entidades>
        <nombreEntidad>METAL</nombreEntidad>
    </entidades>
    <entidades>
        <nombreEntidad>ACCESORIOS</nombreEntidad>
    </entidades>
     </comunicacion>
  </Entidad>

  ...

</configuracion>
```

En donde el nivel de tolerancia que el sistema distribuido requiere es 3 y la definición de comunicación entre sus módulos muestra que el servidor administrador interactúa con el servidor de mapa, con los cuatro servidores de rutas y con los servidores de motor, chasis, llantas, metal y accesorios.

Por lo tanto, el sistema activo **adapta** su comportamiento dependiendo de los parámetros que el sistema distribuido establece en el archivo de configuración.

En la configuración, el sistema distribuido proporciona al sistema activo las interfaces de comunicación entre sus módulos, es decir, las clases ***AccesoriosInterfaz***, ***AdministradorInterfaz***, ***ChasisInterfaz***, ***LlantasInterfaz***, ***MapaInterfaz***, ***MetalInterfaz***, ***MotorInterfaz***, ***RutaCuadrante1Interfaz***, ***RutaCuadrante2 Interfaz***, ***RutaCuadrante3Interfaz***, y ***RutaCuadrante4Interfaz*** son proporcionadas en el proceso de configuración. El apéndice B presenta los diagramas de éstas clases implementadas por el sistema distribuido de transporte.

Las figuras 6.6 y 6.7 presentan los procesos de integración del sistema activo de tolerancia a fallas al sistema distribuido de transporte desde cada una de las partes.

La figura 6.6 muestra el sistema distribuido de transporte en el proceso de configuración del sistema activo de tolerancia a fallas y la suscripción y notificación de inicio del servidor Administrador al mismo.

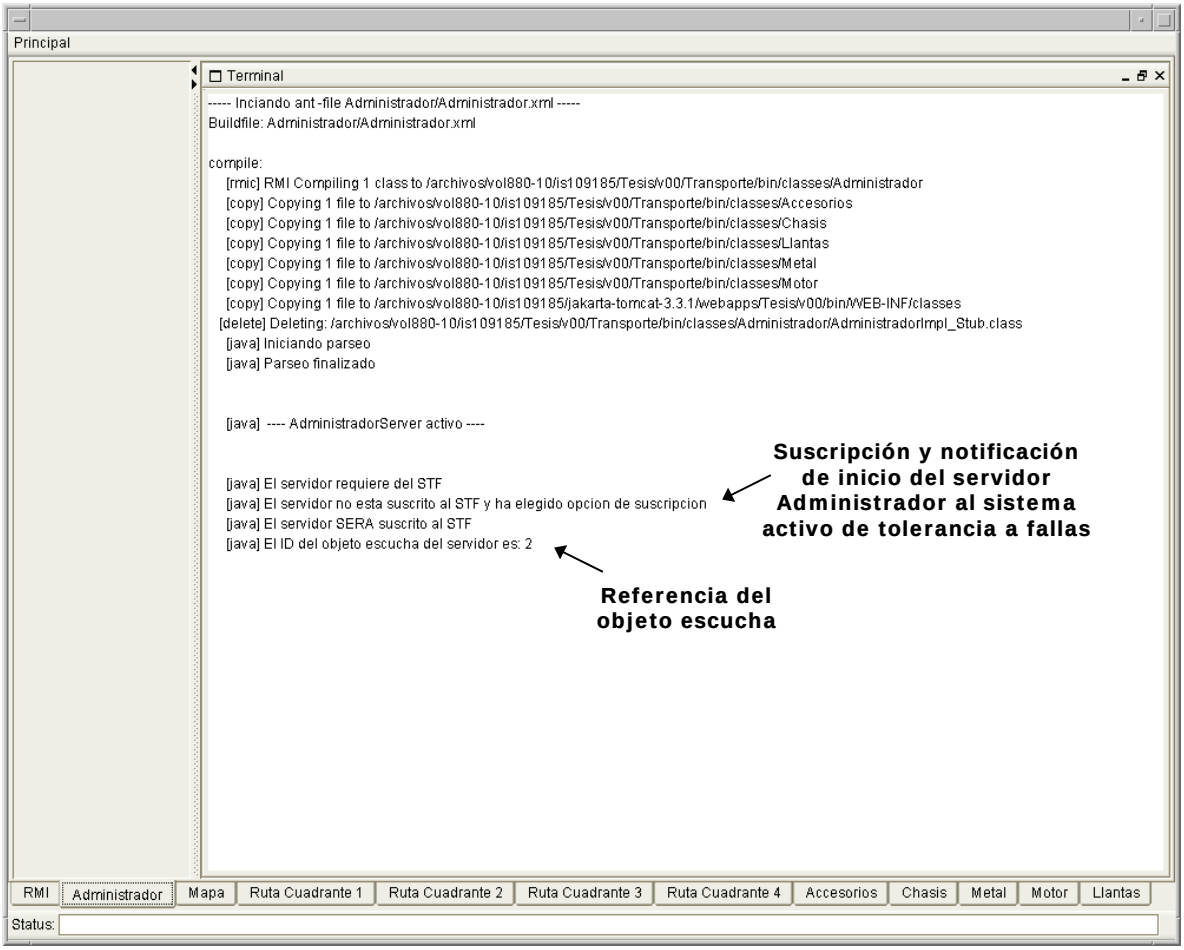



*Figura 6.6 Servidor administrador del sistema distribuido de transporte suscrito al sistema activo de tolerancia a fallas*

Los módulos que conforman el sistema distribuido de transporte se suscriben, enviando el archivo XML que define los tipos de eventos a monitorear y los parámetros que definen sus condiciones correspondientes, y notifican su inicio al sistema activo de tolerancia a fallas de manera remota obteniendo la referencia de su objeto escucha.

Una vez que un módulo del sistema distribuido de transporte notifica su inicio, el sistema activo de tolerancia a fallas adaptado se **integra** para monitorear sus eventos, diagnosticar fallas y corregirlas si es requerido.

La figura 6.7 muestra el sistema activo de tolerancia a fallas en el proceso de configuración, suscripción y notificación de inicio de los módulos del sistema distribuido de transporte. Para cada módulo suscrito, el sistema activo de tolerancia crea los objetos escucha y las *queues* de mensajes correspondientes. Cuando los módulos notifican su inicio, el sistema inicia los objetos escucha asociados para la recepción de sus mensajes.

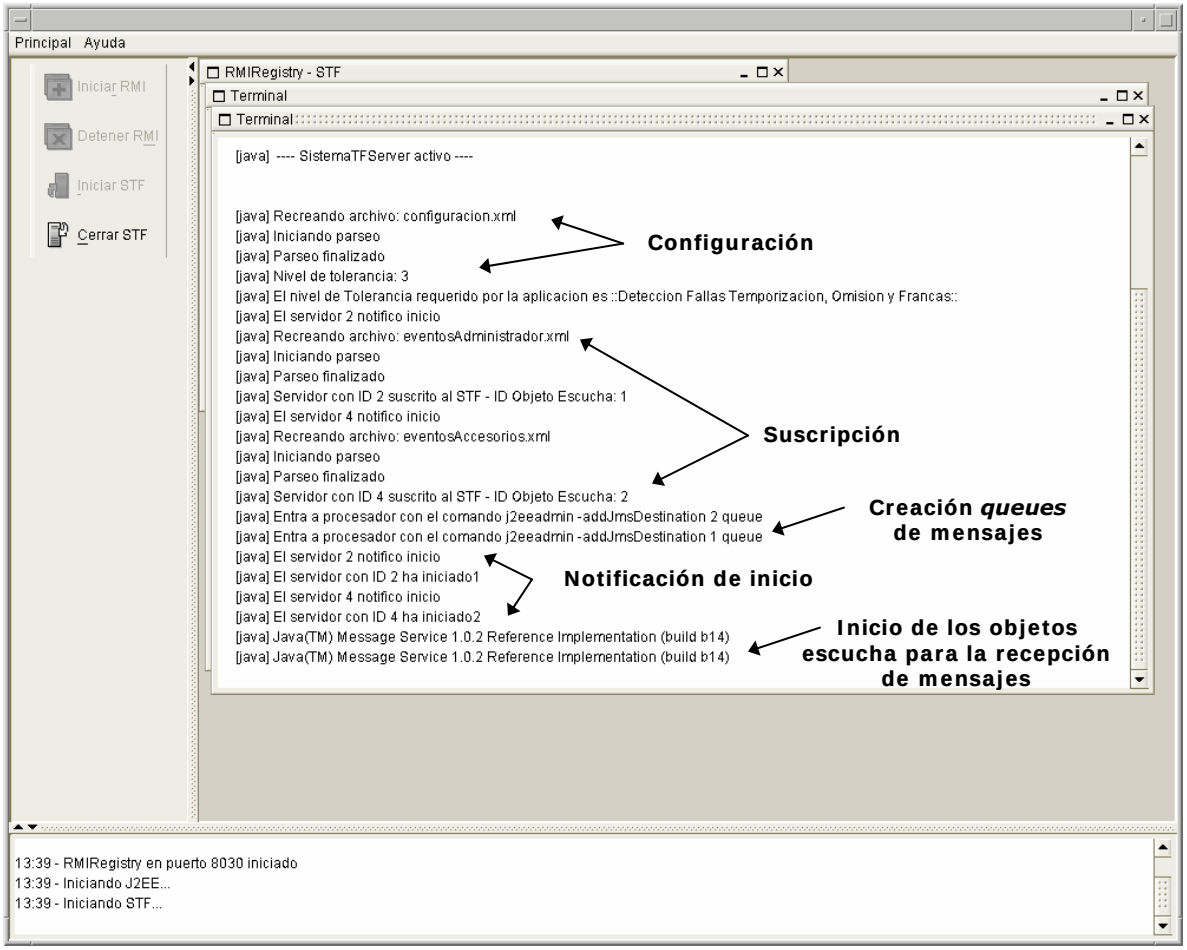



*Figura 6.7 Sistema activo de tolerancia a fallas configurado por el sistema distribuido de transporte con suscripción y notificación de inicio de los servidores Administrador y Accesorios*

Para cada módulo de la aplicación suscrito al sistema de tolerancia a fallas es necesario un enviador de mensajes. La clase ***EnviadorMensajes***, descrita en el apéndice B, implementa el mecanismo de comunicación de un módulo de la aplicación con el JMS.

Por lo tanto, los módulos de la aplicación deben enviar como mensaje el método a través de la instancia a la clase ***EnviadorMensajes*** para cada uno de los métodos de comunicación a ser verificados entre los módulos. La clase ***Metodo***, descrita en el apéndice B, implementa el objeto que representa un método de comunicación entre módulos, junto con sus parámetros. La clase ***Parametro***, descrita en el apéndice B, implementa el objeto parámetro de un método, su posición, tipo y valor.

Para el método de *compra(nombreProducto, cantidad, idProveedor, entidadOrigen, entidadDestino)* el proceso de envío de un mensaje es el siguiente:

```
// Enviador de mensajes
EnviadorMensajes em = new EnviadorMensajes(idObjetoEscucha);

// Creación del método
Metodo metodo = new Metodo();
```

```
Parametros param;
Vector parametros = new Vector();
String IdMetodo = "compraEP";
String nombreMetodo = "compra";

// Parametro nombre del producto
param = new Parametros();
int posicion = 1;
String tipo = "String";
String valor = nombreProducto;
param.setPosicion(posicion);
param.setTipo(tipo);
param.setValor(valor);
// Agrega al vector de parametros
parametros.add(param);

// Parametro cantidad
param = new Parametros();
posicion = 2;
tipo = "int";
valor = (new Integer(cantidad)).toString();
param.setPosicion(posicion);
param.setTipo(tipo);
param.setValor(valor);
// Agrega al vector de parametros
parametros.add(param);

// Parametro proveedor
param = new Parametros();
posicion = 3;
tipo = "String";
valor = idProveedor;
param.setPosicion(posicion);
param.setTipo(tipo);
param.setValor(valor);
// Agrega al vector de parametros
parametros.add(param);

metodo.setID(IdMetodo);
metodo.setNombre(nombreMetodo);
metodo.setParametros(parametros);
metodo.setEntidadOrigen(entidadOrigen);
metodo.setEntidadDestino(entidadDestino);

// Envío del mensaje
em.envio(metodo);
```

El objeto método es creado para el método de solicitud de compra de un producto de una entidad (módulo de la aplicación) a otra junto con sus parámetros y es enviado a través del enviador de mensajes a la cola de mensajes correspondiente al módulo que produce el evento.

Finalmente, es importante mencionar que es necesario que el sistema distribuido de transporte pueda cumplir, en las medidas de sus posibilidades, con la especificación de su servicio sin que el sistema activo de tolerancia a fallas se encuentre activo.

Por lo tanto, para el sistema distribuido de transporte se ha implementado un *switch* que permite, si el sistema activo de tolerancia a fallas está activo, realizar las funciones necesarias de los módulos de la aplicación que requieren ser tolerantes con respecto al

sistema activo de tolerancia; de manera contraria, el sistema distribuido de transporte funciona de manera ordinaria (sin tolerancia a fallas).

## 6.4 Pruebas

Una vez implementados el sistema activo de tolerancia a fallas y el sistema distribuido de transporte junto con los requerimientos mínimos establecidos para su integración, las pruebas realizadas en base a las 4 versiones permiten comparar el funcionamiento del sistema de transporte sin tolerancia a fallas y con tolerancia a fallas a diferentes niveles y formas de percepción.

Para la realización de las pruebas del sistema activo de tolerancia a fallas, las fallas ocurridas en el sistema distribuido de transporte fueron inducidas a través de mecanismos aleatorios de inactividad (espera) por un tiempo determinado en la ejecución de los métodos de comunicación entre servidores. La figura 6.8 muestra el proceso de inactividad del servidor Accesorios.

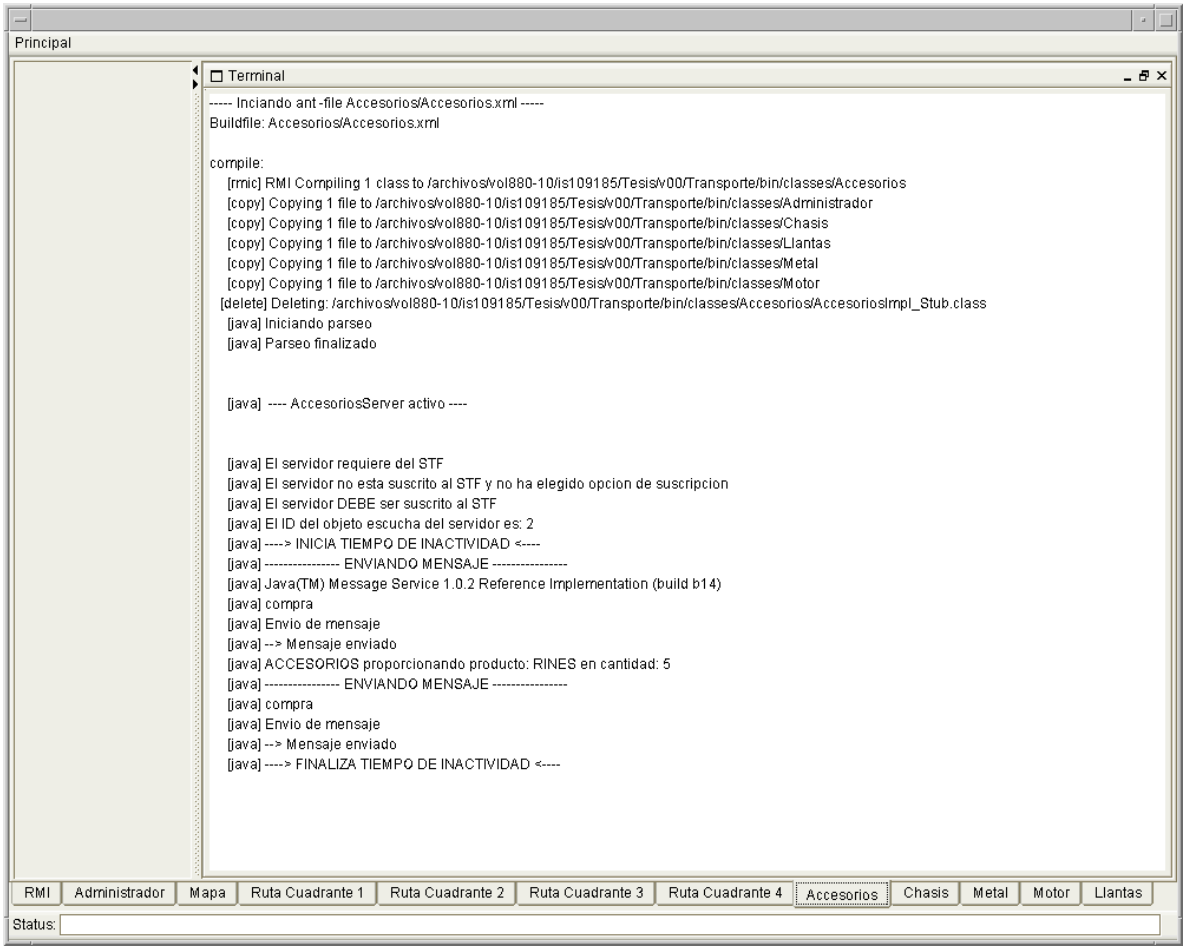



*Figura 6.8 Proceso de inactividad del servidor de Accesorios en la fase de pruebas*

El proceso de inactividad para el servidor de Accesorios fue establecido con una duración de 15 segundos, permitiendo la detección de fallas de temporización, cuyo tiempo de espera de este servidor para la recepción de los eventos que conforman una petición fue definido de 10 segundos.

En la versión 0 el sistema activo de tolerancia a fallas, a través del monitoreo de las acciones del sistema distribuido de transporte, diagnostica fallas de temporización, omisión y francas y las señala a través de la impresión en pantalla de su ocurrencia.

En la versión 1 el sistema activo de tolerancia a fallas diagnostica fallas de temporización, omisión y francas, corrigiendo las fallas de temporización y señalado las fallas de omisión y francas.

La figura 6.9 muestra los procesos de detección y corrección de fallas que el sistema activo de tolerancia a fallas realiza en la versión 1.

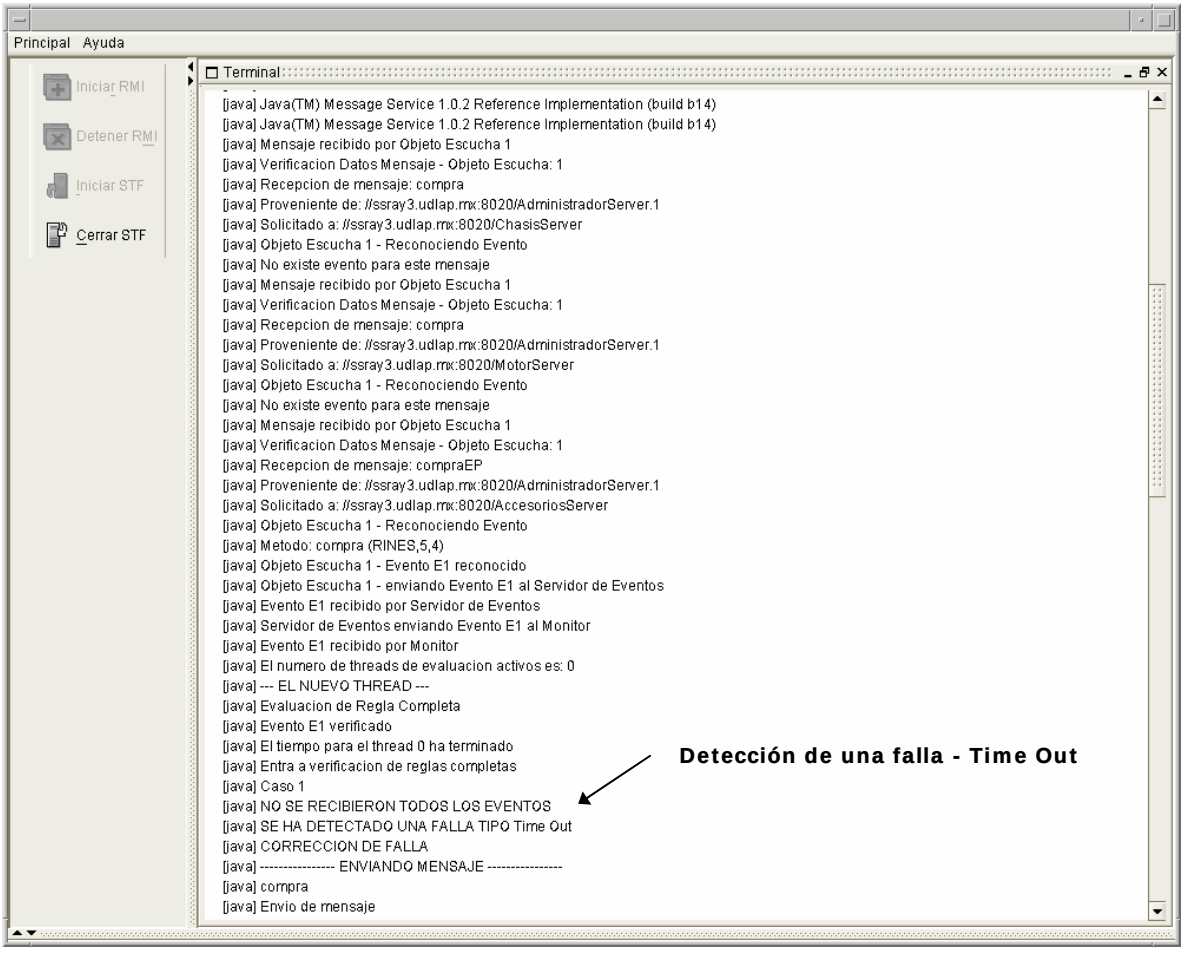



*Figura 6.9 Procesos de detección y corrección de una falla de temporización – versión 1*


*Figura 6.9 Procesos de detección y corrección de una falla de temporización – versión 1 (continuación)*

En la versión 2 el sistema activo diagnostica fallas de temporización, omisión y francas, corrigiendo las fallas de temporización y omisión, y señalado las fallas francas.

En la versión 3 el sistema activo de tolerancia a fallas diagnostica fallas de temporización, omisión y francas y las corrige.

La figura 6.10 presenta el sistema distribuido de transporte con tolerancia a fallas funcionando de manera exitosa, por lo tanto, ante la ocurrencia y corrección de una falla de temporización, omisión o franca, el sistema distribuido satisface el servicio de ensamble de un automóvil dado en la solicitud de compra de un usuario.

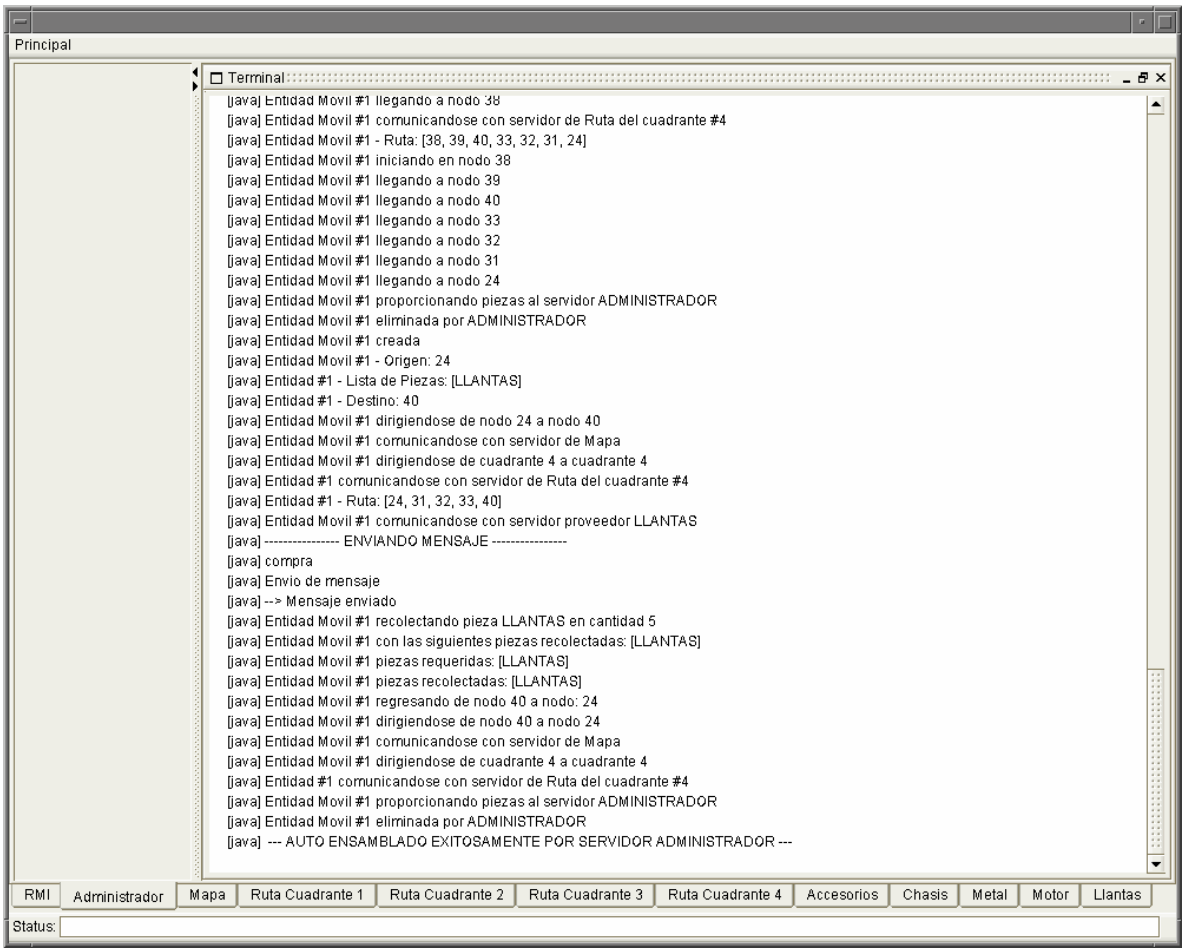



*Figura 6.10 Sistema distribuido de transporte con tolerancia a fallas funcionando exitosamente*

Las pruebas de estas cuatro versiones han demostrado tanto cualitativamente como cuantitativamente la mejora en el funcionamiento del sistema distribuido de transporte.

El proceso de corrección que realiza el sistema activo de tolerancia, ante la detección de la ocurrencia de fallas en el sistema distribuido, es rápido y eficaz evitando la ocurrencia de errores y permitiendo a la aplicación continuar con la especificación de su servicio.

## 6.5 Conclusiones

La implementación de una aplicación distribuida de validación, como lo es el sistema distribuido de transporte, permite la realización de pruebas que comprueben las propiedades y las ventajas de un sistema activo de tolerancia a fallas independiente de la aplicación.

El sistema activo de tolerancia cumple satisfactoriamente los aspectos de adaptabilidad e integración para la aplicación distribuida de transporte implementada. A través del uso de los conceptos de tolerancia a fallas y de los sistemas activos, el sistema activo de tolerancia

a fallas muestra un comportamiento exitoso ante el monitoreo de los eventos producidos por la aplicación para el diagnóstico de fallas y su corrección a través de la estrategia de recuperación establecida.