UNIVERSIDAD POLITÉCNICA DE VALENCIA

Departamento de Informática de Sistemas y Computadores

ARQUITECTURA TOLERANTE A FALLOS MEDIANTE UN SISTEMA MULTIAGENTE PARA EL SISTEMA DE CONTROL DE UN ROBOT MÓVIL

Tesis Doctoral

Presentada por:
Dña. María Guadalupe Alexandres García

Dirigida por:
D. Rafael Ors Carot

Valencia, 2007

Tesis Doctoral

Septiembre de 2007

Para investigar la verdad es preciso dudar, en cuanto sea posible, de todas las cosas

René Descartes

Leonardo Da Vinci

Agradecimientos

El temor de Dios es el principio de la sabiduría, y el conocimiento del Santísimo es la inteligencia
Provervios 9:10

Quiero expresar mi agradecimiento a tre personas que han marcado mi vida que han hecho posible la que me supere día a día en lo personal principalmente y en lo profesional a mi madre, que me inculcó desde pequeña el deseo por el saber, a mi abuela materna Rita que me enseño los valores morales y espirituales que aún conservo y a mi esposo que gran parte de este trabajo se lo debo a él ya que sin su apoyo y comprensón no lo hubiera terminado gracias Luis de verdad.

Agradesco de una manera muy especial a mi director Dr. Rafael Ors Carot que ha sabido darme suficiente libertad para realizar este trabajo y al mismo tiempo aconsejarme para tomar decisiones, y ayudarme a terminarlo de la mejor manera.

Por supuesto, no olvido a las personas que también me han ayudado con su esfuerzo y a quienes les estoy muy agradecida, a mi amiga y mentora la Maestra Leticia Amezaga Ramírez, al Maestro Luis Armando Cheu Ramírez y al Dr. Román Moreno.

Con un cariño inmenso por lo que hemos compartido juntos a mis compañeros del grupo de investigación en el Instituto Tecnológico de Hermosillo, Raúl, Raúl2. Lucero, Juan, Jesús Francisco, Ángel, Luís Carlos, Maria Yolanda, Cecilia Margarita, Luís Alfonso, Alma Liliana y Rito de todos ello he aprendido más que lo que he podido darles como su asesora.

A todos mis compañeros les agradesco el apoyo que me han dado, y de una manera muy especial a Lupita Verduzco, Roció Ramírez, Guadalupe Federico y Claudia García, del departamento de Sistemas Computacionales del Instituto Tecnológico de Hermosillo por los buenos y malos momentos que hemos pasado a lo largo de nuestra enstancia en este centro de trabajo, que nos a dejado muchos gratos momentos cuando vemos a nuestros alumnos crecer como profesionistas.

No me olvido de mis amigas de batalla, las que me han apoyado con su cariño y compañía, les agradesco infinitamente su amistad y su porras para que teminara este trabajo de investigación con cariño a Romelia Martha, Flor, Celin, Irma, Lety, Araceli y Blanca.

Agradesco a mis tres hermanos por su cariño y su comprensión que me han brindado a lo largo de mi vida gracias por que Dios nos ligó en esta vida no pude tener mejores hermanos Irma, Memo y David.

Y de una manera muy especial a la luz de mi vida por la que me mantego de pie a pesar que a veces me doblo, mis 3 hijos y mi nieto por ser mi inspiración, y por el tiempo que no les dì cuando estaba realizando este trabajo: Luis Guillermo, Gustavo David, María de Jesús y Luis Pablo

Resumen

Desde el siglo pasado el ser humano ha perseguido el sueño de construir un robot autónomo y con una gran disponibilidad en su funcionamiento capaz de realizar tareas inteligentes similares a las de los humanos para hacer la vida de estos más sencilla y placentera. Movidos por el afán de construir un robot autónomo, los primeros proyectos de robots móviles datan de los años 70's si bien las técnicas disponibles en aquellos momentos no permitían alcanzar las perspectivas creadas inicialmente. Fue a comienzos de los 80's cuando la investigación en este campo volvió a tomar especial auge. En la actualidad, sin haber alcanzado el objetivo de construir robots móviles personales totalmente autónomos, se han logrado grandes avances en este campo. En la mayor parte se trata de prototipos elaborados por universidades que pasan a ser parte de la industria en robótica para su posterior comercialización. Algunas compañías como Honda que ha destinado grandes sumas a la investigación en este campo actualmente ya ha desarrollado robots muy complejos que están siendo utilizados en la industria automotriz, en el área de la construcción, en la minería, en aeronáutica, en recate, en la agricultura, en el sector médico, exploración en el planeta Marte, etc.

Los desarrollos en robótica indican que en la próxima década existirá toda una nueva industria referente a esta nueva tecnología, ya que los robots serán artículos de uso común tal como ha sucedido actualmente con las computadoras. Esta situación, conlleva a la necesidad de construir robots altamente disponibles y confiables, por lo que es imperativo incluir en su diseño un sistema tolerante a fallos.

Este trabajo se centra en tolerar los fallos a nivel hardware y software en el sistema de control de un robot móvil, de tal manera que la supervisión, detección y recuperación de fallos se implementa independientemente al sistema de control y la plataforma en la que se desarrolla el robot. Esto se logra mediante el diseño de una arquitectura tolerante a fallos implementada con un Sistema Multiagente (MAS). Este sistema lo integran un grupo de agentes encargados de la detección y diagnóstico de fallos.

La arquitectura tolerante a fallos está integrada por dos tipos de agentes principalmente los que se encargan de detectar y recuperar fallos a nivel de software (tareas) y los encargados de tolerar los fallos a nivel hardware (sensores, actuadores, memorias, controladores de red, microcontroladores, etc.). Estos agentes tolerantes a fallos ejecutan los mecanismos tolerantes a fallos de una manera muy simple haciendo acopio de una de sus características que es la intercomunicación y cooperación entre ellos, pudiendo así: detectar, aislar, reconfigurar y tratar de recuperar a un componente ante fallos (a nivel hardware y software) que se presenten durante el funcionamiento de robot.

Para poder desarrollar eficientemente la arquitectura tolerante a fallos propuesta fue necesario modificar la arquitectura de control a nivel software denominada 3+ integrada en el robot, así cómo la arquitectura física (distribuida compuesta por nodos, donde a cada nodo se le conecta como máximo 2 dispositivos ya sea de entrada y/o salida, cada nodo cuenta con un microcontrolador, y sus tareas de control, navegación y planeación).

El SMA que constituye la arquitectura tolerante fallos propuesta, fue diseñada utilizando la metodología MaSE (Multi-Agent Systems Software Engineering) [DeLoach 2001] ya que su contracción esta realizada bajo modelos matemáticos bien definidos. El uso esta metodología para modelar el SMA nos fue muy útil ya que sirvió de guía en el desarrollo de dicho sistema, además nos permitió realizar la transición fácilmente de la fase de análisis a la fase de diseño, basándonos en los requisitos iniciales del sistema de una manera comprobable, eficiente, comprensible, sin ambigüedades y contradicciones.

Se requirió realizar el análisis de las estructuras software utilizado en los robots móviles de tal manera que se seleccionó la denominada 3+ y se implemento el sistema desarrollado en este trabajo Haciendo una modificación en la tercera capa dicha arquitectura software, con la finalidad acoplar nuestro SMA tolerante a fallos, obteniendo un eficiente funcionamiento después de realizar las pruebas pertinentes en el simulador.

Se investiga la teoría de los conjuntos difusos como una herramienta para el diagnóstico del sistema por medio de un árbol de fallos difusos. El objetivo es diagnosticar fallos en componentes por medio de la observación de síntomas difusos usando la información contenida en un árbol de fallos. Para la solución del problema se sigue un procedimiento en dos etapas. En el primer paso, se utiliza el razonamiento causal para diagnosticar los modos de fallo, que consiste en realizar el corte mínimo a los eventos básicos del árbol de fallos, la observación de las puertas accionadas son tratadas como síntomas. En el segundo paso, se identifican los componentes particulares que han fallado basado en los modos de fallo diagnosticados. Para realizar este segundo paso, la solución es mediante una ecuación relacional difusa a = V-producto (alfa x del t de S/sup) conectando el modo de fallo a los eventos básicos derivados de x. Con este método, las ecuaciones del diagnóstico se pueden generar y solucionar simétricamente en términos de eventos básicos del árbol.

Como último paso se diseño un simulador bajo la plataforma de JADE con la finalidad de probar y validar la arquitectura aquí propuesta.

Abstract

From the last century the human being has persecuted the dream to construct an independent robot and with a great availability in his operation able to make intelligent tasks similar to those of the humans to make simpler and placentera the life of these. Moved by the eagerness to construct an independent robot, the first movable projects of robots date from 70 years' s although the techniques available at those moments did not allow to reach the perspective created initially. S was at the beginning of the 80' when the investigation in this field returned to take special height. At the present time, without to have reached the objective to construct robots movable totally independent personal, great advances in this field have been obtained. In most one is prototypes elaborated by universities that happen to be part of the industry in robotics for their later commercialization. Some companies like Honda that has destined great sums to the investigation in this field at the moment already have developed robots very complex which they are being used in the automotive industry, in the area of the construction, the mining, aeronautics, in hides, in agriculture, the medical sector, exploration in the Mars planet, etc.

The developments in robotics indicate that in the next decade all a new industry referring to this new technology will exist, since robots will be articles of common use as it has happened at the moment to the computers. This situation, entails to the necessity to construct robots available and highly reliable, reason why it is imperative to include in his design a fault tolerant system.

This work is centered in tolerating the failures at level hardware and software in the system of control of a movable robot, in such a way that the supervision, detection and recovery of failures independently implement to the system of control and the platform in which the robot is developed. This is obtained by means of the design of a fault tolerant architecture implemented with a System Multiagente (MAS). This system integrates a group of agents in charge of the detection and diagnosis of failures.

The fault tolerant architecture is integrated by two types of agents mainly those that are in charge to detect and to recover failures at software level (tasks) and the ones in charge to tolerate the failures at level hardware (sensorial, actuators, memories, controller of network, microcontroller, etc.). These fault tolerant agents execute the fault tolerant mechanisms of a very simple way making one storing of their characteristics that are the intercommunication and cooperation among them, thus being able: to detect, to isolate, to reshape and to try to recover to a component before failures (at level hardware and software) that appears during the operation of robot.

In order to be able to develop efficiently propose the fault tolerant architecture it was necessary to modify the architecture of control at denominated level software integrated 3+ in the robot, as well as the physical architecture (distributed composed by nodes, where to each node it at the most connects 2 devices to him or of entrance and/or exit, each node counts on a microcontroller, and his tasks of control, navigation and planning).

The SMA that constitutes propose the tolerant architecture failures, was designed using the MaSE methodology (Multi-Agent Systems Software Engineering) [DeLoach 2001] since their contraction this made under defined mathematical models good. The use this methodology to model the SMA was very useful since it served to us as guide in the development of this system, in addition allowed us to easily

make the transition of the phase of analysis to the phase of design, basing to us on the initial requirements of the system of a comprobable, efficient, comprehensible way, without ambiguities and contradictions.

It was required to make the analysis of used the software structures in robots movable in such a way that denominated 3+ were selected and I implement the system developed in this work Doing a modification in the third layer this software architecture, with the purpose of connecting our fault tolerant SMA, obtaining an efficient operation after making the pertinent tests in the simulator.

The theory of the diffuse sets like a tool for the diagnosis of the system by means of a tree of diffuse failures is investigated. The objective is to diagnose failures in components by means of the observation of diffuse symptoms being used the information contained in a tree of failures. For the solution of the problem a procedure in two stages is followed. In the first step, the causal reasoning is used to diagnose the failure ways, that consist of making the minimum cut to the basic events of the tree of failures, the observation of the driven doors are treated like symptoms. In the second step, the particular components are identified that have failed based on the diagnosed ways of failure. In order to make this second step, the solution is by means of diffuse a relational equation to = V-product (alpha x of the t of S/sup) connecting the way of failure to the basic events derived from x. With this method, the equations of the diagnosis can be generated and be solved symmetrically in terms of basic events of the tree.

As last step design a simulator under the platform of JADE with the purpose of proving and of validating the here propose architecture.

CONTENIDOS

Lista de figuras

Lista de tablas

Capítulo 1 Introducción

1.1 Antecedentes

Una de las principales carácterısticas de un ser vivo es la de reaccionar, bien de forma refleja bien de forma deliberada, ante cambios detectados internamente y en su entorno. Los diseñadores de robots, en su aspiración por emular las capacidades biológicas, tratan de incorporar esta funcionalidad con un nivel de competencia comparable. La tarea no es sencilla. Un sistema biológico es capaz de reaccionar de forma adecuada aún en casos de falta de información o en presencia de datos imprecisos, procesando y registrando de forma automática las experiencias pasadas para tratar de mejorar constantemente su rendimiento. La tolerancia a fallos, es por lo tanto, una propiedad igualmente deseable y perseguidas por los sistemas artificiales. Los sistemas robóticos son claros exponentes de estos intentos de emulación [Hernández Daniel, 2003].

El uso de robots en la vida diaria está dejando de ser una ficción. Los actuales desarrollos en el campo de la robótica alrededor del mundo indican que en una o dos décadas existirá toda una nueva industria ya que los robots serán artículos de uso común tal como ha sucedido con el uso de las computadoras. El surgimiento de los robots implica que deben operar en ambientes desconocidos, que normalmente son muy complejos y dinámicos. Esto trae consigo importantes retos de razonamiento con incertidumbre, planeación, coordinación entre robots, comunicación entre los humanos y el robot, así como el tolerar los fallos para lograr que los robots sean verdaderamente confiables cuando requerimos usarlos en tareas donde está de por medio salvar vidas humanas como podría ser: en rescate de siniestros naturales o causados por los seres humanos, intervenciones médicas con alto grado de precisión, desarme de bombas, manejo de material radioactivo, etc.

A partir de esto surge la idea de desarrollar una arquitectura tolerante a fallos a nivel Software y Hardware para el sistema de control de un robot móvil por parte del Grupo de Tolerancia a Fallos, del departamento de informática de sistemas y computadores (DISCA) en la Universidad Politécnica de Valencia, formada por un grupo interdisciplinario de investigadores cuyo fin es lograr avances significativos en ésta área y contribuir al desarrollo de robots verdaderamente confiable. El grupo de investigación antes mencionado tiene como finalidad contribuir en el desarrollo de robots primordialmente de servicio.

1.1.1 Los Robots de la última generación

En la actualidad no existe una definición universal de robot de servicio. A la espera de un acuerdo sobre la misma, la Federación Internacional de Robótica (Internacional Federation of Robotics IFR) ha adoptado la siguiente definición provisional: Robot de servicio es un robot que opera de forma parcial o totalmente autónoma para realizar servicios útiles para el bienestar de los humanos o sistemas de automatización, excluyendo operaciones de manufactura. Los robot de servicio tienen una relación muy estrecha con los seres humanos, como es vigilarlos, rescatarlos, protegerlos, por tal motivo el hombre depende de la operación del robot de una manera directa, de tal manera que uno de los objetivos primordiales es diseñarlos con mecanismos que puedan tolerar los fallos de una manera transparente para los seres humanos que los utilizan. Los robots de servicio son clasificados de la siguiente manera:

- ***Servicio a humanos.***
  - Utilización de robots en la casa para apoyo en labores domésticas, protección a los seres humanos, entretenimiento, etc.
- ***Servicio a sistemas de automatización.***
  - Mantenimiento, reparación, limpieza, etc.
- ***Otras funciones autónomas.***
  - Apoyo en labores de vigilancia, transporte, adquisición de datos etc.

Un ejemplo de los robots de servicio son los robots miniatura llamados *marsupiales* (figura 1.1) siendo los primeros en participar en las tareas de rescate de personas que se encontraban debajo de los escombros en las torres del World Trade Center el 11 de septiembre del 2001, estos robots se deslizan por lugares donde los humano, perros de rescate o máquinas no pueden llegar. Son el producto del trabajo de los alumnos en la University of South Florida comandados por su profesora Robin Murphy (figura 1.2) en el Center for Robot Assisted Search & Rescue (CRASAR).

Figura 1.1 robot marsupiales utilizados en actividades urbanas de la búsqueda de rscate(USAR) en las repercusiones del ataque del WTC el 11 de septiembre del 2001.

Estos robots ayudaron a encontrar ciento veinticinco víctimas y muchos restos humanos. Actualmente estos robots carecen de cables y se guían mediante control remoto y cuentan con subrutinas basadas en la inteligencia artificial, utilizan un nuevo sensor de visión nocturna que incluye FLIR (impermeabilizado), este sensor fue utilizado por primera vez como soporte a los bomberos que ayudaron en los recates de personas en la ruinas que dejó el huracán Charley en Punta Gorda Florida.

Figura 1.2 Los robots marsupiales usados en la zona cero junto a su diseñadora Robin Murphy.

Este sensor fue desarrollado originalmente por la compañía Ford Motors para ser usado en sus automóviles de tal manera que los usuarios puedan manejar de noche con mayor seguridad. La oficina de investigación naval de los Estados Unidos de Norte América vio que podía utilizarlo en la defensa militar, incluyendo la búsqueda y rescate de sus soldados en los campos de batalla, por lo que en

colaboración con la compañía Ford Motors, se lo prestó a CRASAR para que fuera evaluado y probado por sus investigadores. En el Huracán Charley se realizaron estas pruebas.

La utilización de este sensor en los robots es para lograr determinar si una víctima está muerta o viva (miden el aliento de lejos). Una de las pruebas realizadas con este sensor en un robot, fue el proporcionar agua a un ser humano vía una tubería flexible (figura 1.3). Los robots pueden ayudar al personal de rescate a confirmar el estado de una víctima, y proporcionarles cuidado requerido cuando esta atrapada en un área profunda o dentro de escombros inaccesibles por el personal de rescate.

Figura 1.3 Víctima ayudada por un robot diseñado en CRASAR.

Otro grupo de investigadores en Australia trabaja en área de la minería conjuntamente con los científicos de CRASAR en el diseño de un robot miniatura (figura 1.4) de rescate para poderlo utilizar cuando ocurren desastres en el interior de las minas, este tipo de robot esta siendo operando y entrenando experimentalmente bajo tierra a 15 metros de profundidad en Queensland. El diseño está pensado para llevar a cabo operaciones de rescate, tomando en cuenta los requerimientos necesarios del desastre ocurrido en la mina de Quecreek en julio 25 del 2003 cerca de Somerset Pennsylvania.

Figura 1.4 Robot miniatura que realiza operaciones de rescate en desastres ocurridos en las minas.

El equipo de investigación CRASAR es un equipo de rescate en desastres único en el mundo donde utilizan robots como apoyo en su trabajo. Ellos se han entrenado al participar en el rescate de 400 emergencias donde han usado sus robots sin poner en riesgo al personal de rescate. Debido a la experiencia acumulada han podido mejorarlos y hacerlos cada vez más eficientes obteniendo la delantera en el diseño de robots en este tipo, últimamente han tomado la decisión de implementarles algunos mecanismos tolerantes a fallos para que tengan una mayor disponibilidad en su operación.

Otra de las finalidades de construir robots es que puedan intervenir en procesos de manufactura, emergiendo así los robots industriales (figura 1.5). Estos robots, que no tienen forma humana en absoluto, son los encargados de realizar trabajos repetitivos en las cadenas de proceso de fabricación, como por ejemplo: pintar, moldear, soldar carrocerías de automóvil, trasladar materiales, etc. Una fábrica sin este tipo de robots en la actualidad se puede decir que esta fuera de competencia

internacional, los trabajos que realizan actualmente los robots en este tipo de fábricas lo realizaban técnicos especialistas en las líneas de producción. Con la intervención de los robots, el técnico puede librarse de la rutina y el riesgo que sus labores implican, de tal manera que la empresa gana en rapidez, calidad y precisión. Este tipo de robot cuenta con redundancia analítica y física en sus sensores para tolerar fallos.

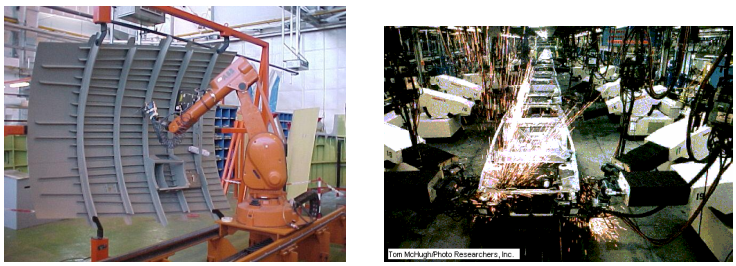


Figura 1.5 Robots industriales.

Por otro lado el laboratorio de inteligencia artificial del Massachussets Intitute of Technology (MIT) donde su director es Rodney Brooks (figura 1.6 arriba izquierda) ha estado trabajando en el diseño de robots inteligentes desde hace tres décadas. El director del mencionado laboratorio se ha caracterizado por realizar investigación estudiando cómo funcionan los seres humanos y algunos animales para poder diseñar máquinas inteligentes, ha diseñado y construido robots que perciben el ambiente que los rodea, entienden nuestro lenguaje, tienen sentido común, aprenden, y actúan en el mundo en que vivimos ayudándonos a desarrollar nuestras labores.

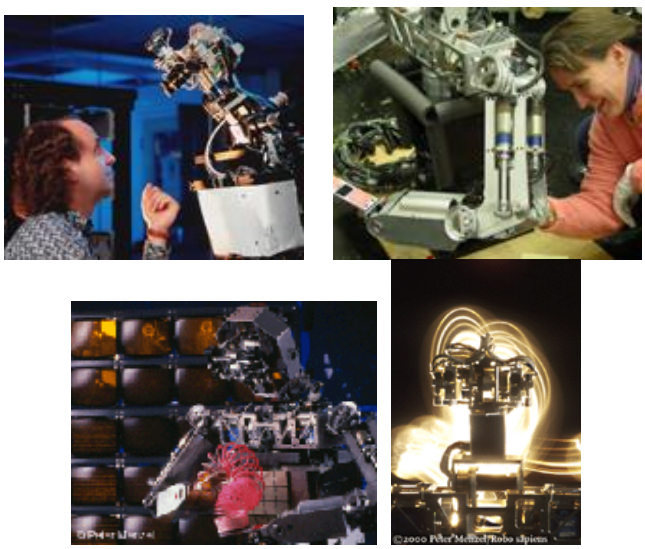


Figura 1.6 Diseño de robots de servicio junto con su creador Rodney Brooks director del laboratorio de inteligencia artificial en el MIT.

El *robot Cong* (figura 1.7) esta en proceso de experimentación tiene forma humanoide, la meta es obtener un robot estético con formas curvilíneas y que tenga una lengua orgánicas esto se está logrando a través de procesos y de materiales avanzados, este robot actualmente se puede mover en su ambiente casi perfectamente ya que cuenta con sensores táctiles y visuales muy sofisticados y con un software diseñado en el mismo laboratorio del MIT que lo controla eficientemente utilizando técnicas de inteligencia artificial vanguardistas. A este robot se le están implementando movimientos a través de la ejecución de sus extremidades tomando en cuenta el metabolismo energético que proveen los músculos de energía, este robot cuenta muy pocos mecanismos tolerantes a fallos.

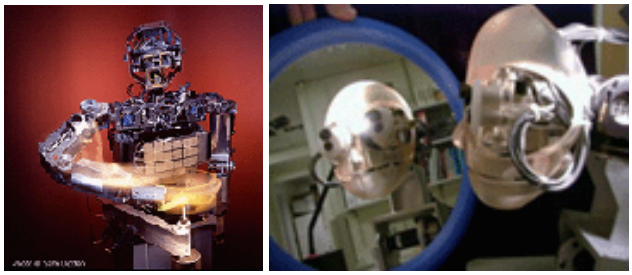


Figura 1.7 Robot Cong humanoide diseñado y construido en el laboratorio de inteligencia artificial del MIT.

Otro robot diseñado en laboratorio de inteligencia artificial del MIT es el proyecto llamado diseño de máquinas sociables es el robot antropomórfico expresivo llamado *Kismet (*figuras 1.8), con este robot se desea probar las expresiones faciales humanas, la posición del cuerpo, la dirección de la mirada, y la voz. Esta inspirado en las expresiones faciales que tienen los niños dependiendo de los diferentes estados emocionales en los que suelen encontrarse, en el robot se integra teorías y conceptos del desarrollo social de los seres humanos, la psicología, la etología, y la evolución infantiles permitiendo a *Kismet*. No tiene integrado en su sistema de control ningún mecanismo tolerante a fallos.

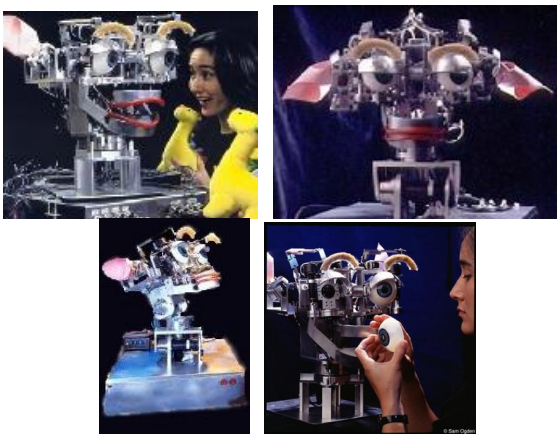


Figura 1.8 Robot Kismet diseñado y construido en el laboratorio de inteligencia artificial del MI.T

El robot Hannibal, y su gemelo Atila diseñados y construidos en el Laboratorio de Robótica Móvil del MIT (Angle 1991). Hannibal es un robot autónomo pequeño con gran rendimiento que lleva a bordo su propio sistema computacional y sensorial. Este robot es quizás el robot más sofisticado y complejo para su tamaño. Una fotografía de Hannibal se muestra en (figura 1.9).

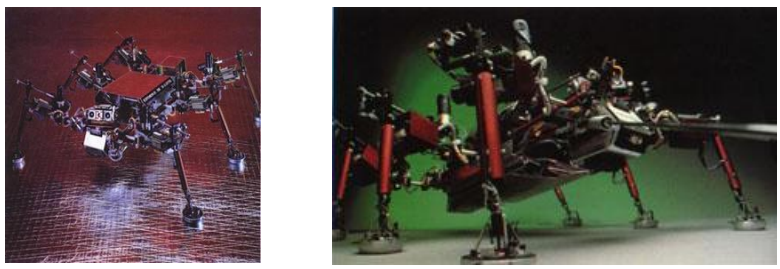


Figura 1.9 Robot Hanibal.

Hannibal se diseñó con patas en lugar de ruedas para lograr mayor movilidad sobre el terreno escabroso, Este robot es el primero que se le implementan mecanismos tolerantes a fallos sofisticados en el mecanismo de locomoción de sus patas [Farell Cinthya 2003].

Por otro lado la agencia espacial de Estados Unidos (Nacional Aeronautic and Spacial Agency NASA), ha diseñado dos robots exploradores para enviarlos a Marte Los robots son exploradores denominados Spirit y Opportunity (figura 1.10). Estos robots han transmitido varios miles de fotografías, además de realizar estudios geológicos que han ayudado a determinar el origen del planeta y de su composición, en una de las misiones de mayor éxito científico de la NASA. *Spirit* tuvo un problema de *software* al llegando a Marte, y a punto estuvo de provocar pánico entre los científicos y los responsables de control de la misión, este se logró arreglar. En el caso de *Opportunity* se atascó en una duda de arena y permaneció varias semanas atrapado hasta que logró salir a un terreno más duro. Esto hizo recapacitar a los científicos que debían realizar mayor investigación en técnicas tolerantes a fallos para implementarlas en los nuevos diseños de próximos robots y así lograr una mayor fiabilidad.

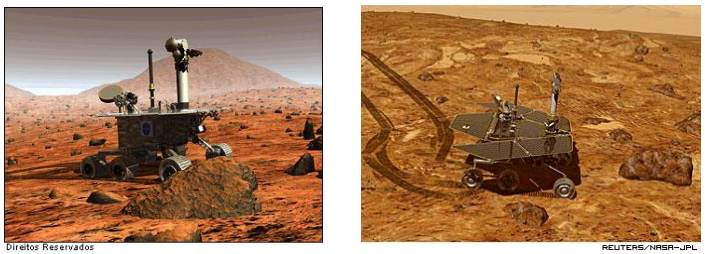


Figura 1.10 Los robot exploradores de Marte Spirit y Opportunity llegaron en enero del 2004, y funcionan como geólogos de campo.

Homayoon Kazerooni profesor de Ingeniería Industrial y director del laboratorio de robótica e ingeniería humana de la Universidad de Berkeley, actualmente está trabajando en el desarrollo de un esqueleto robótico (Lower Extremity Robotic Exoskeleton BLEEX), éste esqueleto ayuda a las personas a caminar perfectamente a través de un terreno áspero y con obstrucciones, además le apoya para subir escaleras fácilmente, ponerse en cuclillas, doblarse y girar de lado a lado sin reducciones sensibles de en su agilidad, llevando a cuestas un paquete con un peso de 40 kilos, esto se logra debido a que el diseño es ergonómico, altamente maniobrable y técnicamente robusto, ya que se combina, el sistema de control humano con el músculo robótico(figura 1.11).

Esta tecnología es fundamentalmente desarrollada para poder reforzar a las personas con capacidad limitada en los músculos inferiores y de esta manera puedan caminar de manera óptima

La máquina puede convertirse también en una herramienta valiosa para cualquier persona que requiere recorrer a pie largas distancias y llevar a cuesta una carga pesada. El esqueleto robótico puede igualmente ser utilizado por los médicos del ejército para cargar a los soldados heridos en el campo de batalla. Asimismo el esqueleto lo pueden utilizar los bomberos para trasladar fácilmente el equipo de contra incendio al área del siniestro, y de igual forma subir rápidamente las escaleras; ya sea para poder entrar donde se localiza el fuego y extinguirse lo antes posible ó para traer alimento y primeros auxilios a las áreas donde los vehículos no pueden entrar [Kazerooni H. 2006].

Figura 1.1 1 esqueleto robótico

Los robots tipo humanoide diseñados por científicos de la Universidad de Tokio son actualmente los mas avanzados, algunos pueden servir té y luego lavar la taza, pero todavía necesitan aprender de sus errores para poder lograr convertirse en eficaces ayudantes en el servicio de la casa. En la figura 1.12 se puede apreciar al robot HP-2 durante una demostración en el Instituto Nacional de Ciencia Industrial Avanzada y Tecnología (AIST), en Tsukuba, Japón. El robot HP-2 simula a un trabajador en una línea de ensamble que es capaz de interactuar con sus compañeros de trabajo, asimismo puede caminar sobre terreno áspero, levantarse después de caer, levantar las canastillas de material, trasladarse al sitio indicado, ponerse en cuclillas, doblarse para recoger algo y girar sobre su eje, el robot mide 1.50 Mts. de altura.

Figura 1.12 Robot HP-2

El primero de Junio del 2007 los investigadores de la escuela de graduados de Ingeniería en la Universidad de Osaka Japón dan a conocer el robot CB-2 (Baby humanoid robot), que actúa como un niño, el diseño del robot tiene como objetivo ayudar a entender mejor el proceso de desarrollo del infante. El comportamiento actual del robot *CB2 baby* (Figura 1.13) es actuar como un niño de entre 1 a 2 años, que pone atención de lo que pasa a su alrededor, se retuerce sobre el piso, y juega en su cuarto con la gracia de un infante de su edad. Las características del CB-2 son: mide 1.30 Mts. de altura, pesa 33Kgs., 56 cilindros de aire le sirven como músculos, usa cámaras fotográficas para los ojos, micrófonos para los oídos, y 197 sensores táctiles ensamblados bajo una capa del silicón suave que le cubre todo su cuerpo, el CB2 está bien equipado para soportar estímulos ambientales. Cuando se le golpea ligeramente sobre sus hombros realiza una mueca en su cara como si estuviera sorprendido entonces voltea su mirada y la fija hacia la persona que lo tocó, cuando un juguete se localiza delante de sus ojos, dedica toda su energía para lograr alcanzarlo. El CB2 también posee un sistema para sus cuerdas vocales artificiales que utiliza para hablar como un bebé.

Figura 1.13 Robot CB2 Baby.

Los investigadores que diseñaron el CB2 están actualmente trabajando para equipar al robot del software que le de la capacidad de aprender, y el desafío a largo plazo es enseñarle cómo caminar y hablar. A pesar que el robot es muy avanzado en técnicas de: mecánica, electrónica y en inteligencia artificial su diseño contempla casi nada la tolerancia a fallos.

Geminoid es un robot teledirigido, diseñado por el profesor Hiroshi Ishiguro en la Universidad de Osaka es investigador en los laboratorios inteligentes de robótica y de comunicación.

El cuerpo del robot es una copia del profesor Hiroshi Ishiguro (figura 1.14), la forma del cráneo de Geminoid fue creada basada en las exploraciones de MRI de la cabeza del profesor H. Ishiguro. El robot comparte algunos de los ademanes del profesor. El cuerpo de Geminoid, fue producido por Kokoro, los fabricantes de la línea de Actroid de fembots, tiene 46 grados de libertad y es conducido por un sistema de compresores de aire. La piel tiene una gran suavidad debido al material utilizado que es caucho de silicón. Por el momento el robot está confinado a una silla, no puede estar de pie ni caminar. Los cables de comunicación y de transmisión salen de un extremo posterior de la silla. Desarrollar el cuerpo duró un periodo de tiempo de 6 meses y de 2 a 3 meses fueron utilizados para desarrollar el software. Uno de los propósitos de crear al Robot Geminoid es explorar el concepto de la tele-existencia, esto es copiar la *presencia* de un ser humano real de modo que él o ella puedan existir en dos lugares a la vez. Éste robot no contempla en su diseño mecanismos tolerantes a fallos.

Figura 1.14 A la izquierda el robot Geminoid con su gemelo humano y a la derecha sentado en su silla.

En 1999 la compañía Sony lanza al mercado la generación de robots mascota con forma de perro (figura 1.15) con el nombre de AIBO en sus diferentes modelos, en referencia al uso de de la inteligencia artificial. Están provistos de motores que activan sus miembros; cuentan con 20GdL repartidos por su cabeza, boca, patas orejas y cola, además cuentan con dos micrófonos para oír, un altavoz para hablar, y una cámara CCD para ver, y otros tipos más de sensores. El comportamiento del robot está condicionado al programa grabado en un memory-stick alojado en su panza. Cuando Sony suministra el robot, éste trae consigo un memory Stick programado con el software diseñado especial para que la

mascota reconozca hasta 100 instrucciones diferentes que le da su dueño, además recrea la educación de la mascota. Mediante una secuencia de instrucciones y la interacción continua, el dueño emula las fases de crecimiento que van desde cachorro hasta el animal adulto El software también instruye al perro para que se comporte como una mascota con diferentes estados de ánimo (alegre, enojado, triste, etc.), reconoce también algunas órdenes por voz, cuenta con un sistema de visión artificial, etc. No obstante, este programa puede ser cambiado de varias formas. Las principales son dos: la primera es a través del sistema R- Code, que consiste en insertar un memory stick con un nuevo programa escrito en el lenguaje C y luego compilado; la segunda consiste enviarle comandos de movimiento a través de la red wifi, desde la PC que ejecuta un programa, normalmente escrito en C. Este segundo método se conoce como Remote FrameWork (RFW).

Figura 1.15 el robot mascota diseñado por Sony

El máximo logro de la compañía Sony fue el robot humanoide QRIO, tiene 60 cms. de altura, y capacidad de caminar, correr y reconocer voz y el rostro de una persona. Dispone de una tecnología denominada *Intelligent Servo actuator* que es lo que le permite andar dinámicamente (variando r.p.m. y torque en las articulaciones) y emplea una técnica denominada *Zero Moment Point* para mantener la estabilidad. Los datos técnicos están contenidos en la tabla 1.1:

Procesador		***Procesador RISC de 64 Bits (2 unidades)***
Dispositivo de Almacenamiento principal		***64MB DRAM (2 unidades)***
Sistema Operativo		***(S.O. tiempo real de Sony)***
Arquitectura de control del robot		***OPEN-R***
Memoria de programa		***Tarjeta de memoria de 16MB***
Grados de libertad		***Cuello: 4 grados de libertad, Cuerpo: 2 g.d.l., Brazos: 5 g.d.l. (x2), Piernas: 6 g.d.l. (x2); total 28 g.d.l. + 5 dedos en cada mano.***
Sensores internos	***Distancia***	***Infrarrojos: cabeza x1, manos x2, total 3***
	Aceleración	***Tronco: X, Y, Z/3 ejes, piernas: X, Y/2 ejes***
	Inclinación	***Tronco: X, Y, Z/3 ejes***
	Planta del pie	***Sensor de presión (cada pierna: 4 x 2 = total 8)***
	Térmico	***Externo (x4), Interno (x2)***
Sensor de contacto	***Cabeza***	***Goma sensible a la presión***
	Amarre	***Interruptor protegido de contacto***
	Manos	***Interruptor protegido de contacto (x2)***
	Hombros	***Interruptor de tacto (x2)***
Imagen de entrada		***Cámara color CCD , 110,000 píxeles de 1/5 de pulgada (x2)***
Entrada de sonido		***Micrófono (x7)***
Salida de sonido		***Altavoz***
Entrada/Salida		***Puerto para tarjeta de PC (Tipo II) (x1)***

		Ranura para tarjeta de memoria (x1)
Display	*LED ojo*	*Color 4096 (según RGB 16)*
	LED oído	*Color 1 (grabación 16)*
	LED de energía	*2 colores (3 colores en iluminación simultanea)*
Velocidad andando		*Aproximadamente 6metros/minuto máximo (sobre superficie irregular) Zancada: 10cm, Ciclo de zancada: 1.0 segundos/paso*
		Aproximadamente 20metros/minute máximo (superficie lisa y plana) Zancada: 6.5cm, Ciclo de zancada: 0.20 segundos/paso
Capacidad de andar sobre superficie irregular		*Irregularidad: 10mm de superficie irregular en suelo no resbaladiza*
		Grado máximo de pendiente: Aproximadamente 10 grados sobre superficie no resbaladiza.
Peso		*Aproximadamente 6.5Kg con batería y memoria*
	Dimensiones (altura x ancho x largo)	*Aproximadamente 580 x 260 x 190mm*

Tabla 1.1 Datos técnicos del robot QRIO diseñado por la compañía Sony.

Figura 1.16 Robot QRIO en su rutina de baile

En el 2006 llegó a coordinar a este tipo de robot (figura 1.16) con una rutina de baile cuya coreografía tomó varias semanas de trabajo. La peculiar danza robótica se anima por medio de una PC. QRIO tiene una gran capacidad de manejar un balón de futbol, de tal modo que sus realizadores logro enfrentarlos en un partido de éste juego, con todos estos éxitos los QRIO se quedaron en meros prototipos y nunca salieron al mercado.

En el año 2006 Sony informó que suspendería su programa de robótica, dejaría de producir AIBO y nunca lanzaría al mercado a los QRIO. Sin embargo, aprovecharía algunas de las tecnologías creadas para éstos robots en sus nuevos productos comerciales.

1.1.2 Agentes y Sistema Multiagente

Durante muchos años la investigación en Inteligencia Artificial (IA) se ha orientado hacia aplicaciones independientes con un determinado conocimiento y con un objetivo específico, esta metodología no ha sido suficiente debido a la presencia de un número de entidades que cooperan en el mundo real. En la actualidad la compleja tecnología de la información y los problemas que esta genera, demanda que se desarrollen nuevos enfoques con la habilidad necesaria para contener información abundante,

imprecisa, incompleta y hasta contradictoria; que permitan interactuar con ambientes en los que se requiere competir por los recursos, con suficiente portabilidad para operar sobre diferentes plataformas computacionales [Camacho D., D. Borrajo, J. M. Cammarata S., McArthur D., Steeb R. 2004].

La Inteligencia Artificial Distribuida (IAD) pertenece a la Inteligencia Artificial, cuya finalidad es construir un modelo integrado por entidades inteligentes que interactúan cooperando o compitiendo inmersas en ambiente distribuido y se les conoce como agente de software.

Sus aplicaciones de este nuevo enfoque denominado agente de software son:

- Control en tráfico aéreo.
- Educación.
- Asistencia personal.
- Minería de datos.
- Comercio electrónico.
- Recuperación de la información.
- Sistemas de control.
- Administración de redes para computadoras.
- Análisis de imágenes.
- Robótica.
- Monitorización domiciliaria de pacientes con patologías cardiovasculares.
- En la bolsa de valores.
- Subasta electrónica.
- Industria de fabricación.

Los principales centros de investigación y universidades de mayor prestigio que han participado en el desarrollo de la tecnología de agentes de software se expresan a continuación:

- Univesidad Carnegie Mellon.
- Universidad de Stanford.
- Instituto Tecnológico de Massachussets (MIT).
- Universidad de Londres.
- American Telephone and Telegraph (AT&T).
- International Business Machines Corporation (IBM).
- Hewlett Packard (HP).
- Microsoft.
- XEROX.
- Apple.
- Oracle.
- Lotus.

La tecnología de agentes ha cambiado la manera de interactuar con las computadoras, la de conceptualización y construcción de sistemas complejos, este tipo de sistemas están integrados por componentes simples con comunicación e interacción compleja. La interacción entre agentes se basa en conceptos y terminología que comparten para comunicar conocimiento mediante un lenguaje de comunicación permitiéndole también solicitar información y servicios. Se considera un nuevo paradigma de las ciencias computacionales, denominado programación orientada a agentes (Agent Oriented Pogramming AOP), propuesta realizada por [Shoham, 1993], la cual pude ser vista como una especialización de la programación orientada a objetos. El estado de un agente consiste de componentes como son creencias, decisiones, capacidades y obligaciones, por esta razón al estado de un agente se le conoce como su estado mental. El concepto se puede considerar como un desarrollo e implantación de

la arquitectura (creencias, deseos, intenciones, del inglés belief, desire, intention), propuesta en [Rao A., Georgeff M.1992]. Los agentes son controlados por programas, los cuales incluyen primitivas para comunicarse con otros agentes, cada primitiva de comunicación es de un cierto tipo: información, petición, oferta, etc. Existen algunas herramientas para programar agentes como son los siguientes:

- LALO para plataforma Sun.
- Agent K para plataforma Sun.
- Java Agent Templeta (JAT).
- Aglet.
- JAVA.
- JADE.
- AgleTcl.
- Jack Intelligent Agents.
- Prometheus Design Tool (PDT).

La mayor parte de problemas que se requiere resolver usando la tecnología de agentes puede ser resuelta con un solo agente. Así como el concepto de programación descendente, modular y orientada a objetos intenta dividir los problemas en sub-problemas cada vez más sencillos, en el ámbito de los agentes se renuncia a la idea de un agente único. Es decir, lo mismo que en la sociedad humana podemos encontrar las actividades claramente divididas para conseguir un elevado grado de especialización y eficiencia, en el caso de los agentes se tiende a este planteamiento en el que múltiples agentes autónomos interactúan, se coordinan (cooperando, compitiendo o ambas cosas) y todo ello de un forma adaptativa. Por lo que se constituyen sociedades de agentes a las que se les consideran Sistemas Multiagente (SMA).

Los Sistemas Multiagente ofrecen importantes ventajas sobre los sistemas centralizados, de la misma forma que la computación distribuida mejora las prestaciones de la centralizada. Parece lógico que los agentes no actúan solitariamente, sino que se localizan en entornos o plataformas con varios agentes. Cada uno de los agentes del entorno tendrá sus propios objetivos, tomará sus propias decisiones y podrá tener la capacidad de interactuar y comunicarse con el resto de agentes [Giret, Insfrán, Pastor, Cernuzzi 2000]. Los entornos o plataformas que soportan varios agentes es lo que en la literatura se conoce con el nombre de Sistema Multiagente o agencias.

1.1.3 Tolerancia a fallos en robot móviles

La construcción de los robots como los mencionados anteriormente se han diseñados para realizar sus tareas en ambientes peligrosos o inhóspitos para los seres humanos. Por lo antes expuesto se puede deducir, que las investigaciones en este campo de acción se encamina a desarrollar robots autónomos capaces de realizar exploraciones en los diferentes planetas, en océanos, dentro de volcanes, en minas, drenajes profundos, como ayudantes médicos en los quirófano, etc. Lo que realmente se requiere es que el robot realice su trabajo por largos períodos de tiempo sin la necesidad de realizar reparaciones cada vez que un componente llegue a fallar.

En los sistemas robóticos es muy común ver como el hardware falle con frecuencia, y con menor frecuencia es el caso del software. Los fallos en los sensores y actuadores no son de sorprender dado cómo los robots tropiezan continuamente con objetos durante su trayectoria, a menudo los alambres se desconectan debido a vibraciones, los conectores se desajustan mientras que se mueven a través de su ambiente, las tareas se corrompen , el controlador de red falla, etc. Lo deseable es no tener que parar al robot cada vez que se le presenta un fallo. Para lograr que el robot continúe funcionando a pesar de

tener fallos en el hardware o software, y así pueda alcanzar sus metas, es necesario implementar mecanismos tolerantes a fallos cuando se realiza su diseño garantizando así mayor disponibilidad del robot par realizar sus trabajo.

Los robots móviles generalmente tienen integrado un gran número de sensores y actuadores, esto conlleva ventajas y desventajas. Tener múltiples sensores provee al robot con una sensibilidad más confiable y una visión mejor del mundo que lo rodea. Más actuadores, hacen que el robot pueda realizar mejor su trabajo. Sin embargo, más dispositivos también significan que existen más componentes que puedan fallar y subsecuentemente degradar el desempeño del sistema. Los fallos físicos pueden ser atribuidos ya sea a fallos mecánicos, fallos electrónicos o fallos sensoriales. Los cambios sutiles en el estado del robot como puede ser la caída de señal de un sensor también degradan su desempeño. Para realizar cualquier tarea el robot móvil se apoya en el sistema de locomoción, y de esta manera lograr cumplir con sus objetivos a través interactuando con su ambiente, por lo que esta sujeto a repetidos golpes, obstáculos inesperados y grandes esfuerzos. Esto provoca deterioros en el hardware del robot. Por lo que no es de sorprender que tenga fallos en sus componentes a través del tiempo.

Los robots móviles experimentan varios tipos de fallos en sensores y actuadores. Otros tipos de fallos pueden ocurrir, como son los de fallos en el sistema computacional o en el software, aunque este tipo de fallos no ocurren tan frecuentemente. Los fallos más comunes en este tipo de robots son los fallos en los sensores y/o actuadores. Estos fallos tienen una gran variedad de causas. Por ejemplo, los cables de los sensores se rompen debido a la tensión a la que están sujetos por los puntos de movimiento. Los problemas en los montajes de los sensores como es el ángulo insuficiente en su junta de unión en el caso de los potenciómetros que causan caída en la señal. El valor de referencia de los indicadores de esfuerzo también se cae con el paso del tiempo.

La caja de metal de los potenciómetros en los ángulos de unión puede no estar aislada perfectamente del resto del sistema lo cual puede causar lecturas erróneas en los sensores. El segundo tipo más frecuente de fallos son en los actuadores. Los fallos en los actuadores ocurren en los robots por ejemplo: El eje del motor tiene una tendencia a romperse a través del tiempo. A veces las abrazaderas del motor se aflojan haciendo que el motor rote a un estando no permitido. Mientras el motor rota en su montaje, los cables de señal de movimiento del motor se tuercen y eventualmente se rompen debido a la tensión.

Después de estudiar a los diferentes robot móviles diseñados en los mejores laboratorios de investigación, nos percátanos de que la mayoría no cuentan con mecanismos de tolerancia a fallos, por tal motivo nos dimos a la tarea de diseñar una arquitectura tolerante a fallos para el sistema de control de un robot móvil modelada con un Sistema Multiagente (SMA), de tal manera que diferentes tipos de agentes (nodo y tarea) se encargarán de ejecutar mecanismos tolerantes a fallos, y minimizar el impacto del fallo y por consiguiente el desempeño en el sistema de control del robot. Explotando la concurrencia que esta implícita en las tareas que integran el SMA y con la distribución e interacción entre los agentes se logra detectar y compensan los fallos para cada componente de una manera simultánea.

El Sistema Multiagente tolerantes a fallos tiene como finalidad los siguientes objetivos:

- *Tiempo rápido de respuestas a fallos*: la mayoría de robots móviles opera en ambientes peligroso (rescate, desactivar bombas, tratamiento de material radioactivo, en el espacio). Consecuentemente, este tipo de robot debe detectar y remediar fallos rápidamente si no su seguridad puede peligrar. Esto significa que los fallos deben ser detectados y compensados para prevenir el desempeño del sistema y que se degrade hasta un nivel inaceptable.

- *Degradación suave de desempeño:* El desempeño del robot debe degradarse suavemente mientras los fallos se acumulan. Esto requiere que el robot mantenga el más alto nivel posible de desempeño dado el estado funcional del Hardware.
- *Acceso a todos los recursos confiables*: Más sensores y actuadores elevan el desempeño del sistema previendo que estos sean funcionales. A partir de aquí, el robot deberá reincorporar el uso de los componentes reparados.
- *Cobertura de fallos*: El robot puede sufrir una variedad de fallos. Los fallos pueden ser permanentes o transitorios. Algunos fallos tienen un efecto local mientras que otros tienen un efecto global en el desempeño los sensores pueden descalibrarse. Además, el robot deberá ser capaz de recuperarse de las diferentes combinaciones de fallos. Los fallos pueden ocurrir individualmente, concurrentemente con otros fallos, o acumularse a través del tiempo.

1.2 Justificación

Los robots son sistemas computacionales que cada día son más utilizados por los humanos para que los auxilian en los trabajos arduos, riesgoso y lugares inhóspitos donde se pone en riesgo la vida del ser humano.

Los robots son sistemas complejos, que tiene altas probabilidades de que les sucedan fallos. Y esto trae consigo enviar a personal de mantenimiento de manera frecuente a estos tipos ambientes para reparar cada fallo que puede surgir en sus diversos componentes, entonces la ventaja de usar a los robots en la realización de tareas en ambiente de riesgo se pierden rápidamente. Por lo que es necesario que los robots se diseñen con mecanismos tolerantes a fallos con la finalidad de que pueda detectar y adaptarse a los fallos ya sea a nivel de software o hardware, permitiendo de esta manera que los robots continúen trabajando hasta que las reparaciones puedan ser programadas de una manera efectiva. Por tal motivo en este trabajo de investigación se propone modelar una *Arquitectura Tolerante a Fallos a Nivel Hardware y Software Mediante un Sistema Multiagente para el sistema de control de un robot móvil*, de tal manera que los agentes puedan detectar cualquier fallo en un determinado componente que integra al sistema robótico (hardware o software), accionando los algoritmos tolerantes a fallos previamente diseñados. Los agentes que integran dicho Sistema Multiagente se sustenta en el diseño físico de una arquitectura de control distribuida, con la finalidad apoyar a determinados agentes que integran esta red en el mecanismo de configuración para lograr que el sistema robótico tenga una mayor disponibilidad.

Dada la necesidad de integrar estos elementos dentro de una arquitectura de software, y siendo una de las metas principales del Grupo de Investigación Tolerante a Fallos (GITF) del departamento de informática de sistemas y computadores (DISCA) en la Universidad Politécnica de Valencia España, nos dimos a la tarea de diseñar esta capa tolerante a fallos que se pueda adaptar de una manera flexible en la arquitectura software de un robot móvil.

1.3 Objetivos

1.3.1 Objetivo General

El objetivo principal se centra en lograr mayor disponibilidad y fiabilidad de un sistema robótico, para ello se requiere diseñarlo de tal manera que pueda tolerar los fallos a nivel hardware y software.

1.3.2 Objetivos Específicos

Para lograr el diseño del Sistema Multiagente tolerante a fallos se requiere de:

- Realizar una revisión del paradigma de agente y Sistema Multiagente (SMA).
- Analizar las diferentes metodologías de análisis y diseño de los Sistemas Multiagente, de tal manera que podamos seleccionar una metodología formal adecuada, para lograr modelar nuestro Sistema Multiagente, apegado a los requisitos planeados por nosotros.
- Estudiar los sistemas de control distribuidos para robots y seleccionar el más adecuado para modelar nuestra arquitectura tolerante a fallos.
- Modificación del sistema del control distribuido seleccionado, para que los mecanismos de tolerancia a fallos implementados en los agentes sea mas eficiente y confiables,
- Modelar el Sistema Multiagente tolerante a fallos mediante la metodología formal seleccionada.
- Validar el Sistema Multiagente en lo que respecta al diseño del modelo.
- Realizar un análisis de fiabilidad para diseñar los algoritmos de detección de fallos en los agentes, calcular los porcentajes de fallos de los diferentes componentes del sistema y así validar esta arquitectura.

1.4 Alcances y limitaciones

Éste trabajo tiene como finalidad realizar el diseño y la simulación de una arquitectura tolerante a fallos por medio de un Sistema Multiagente, que sea fácil de integrar mediante una interface con la arquitectura software de cualquier tipo de robot que cuente con un sistema de control distribuido. Asimismo se implementará una doble conexión de los sensores y actuadores que integran la arquitectura hardware distribuida del sistema de control del robot, de tal manera que soporte eficientemente los mecanismos tolerantes a fallos implementados en los agentes.

La arquitectura propuesta tienen su origen en las investigaciones y aplicaciones de robótica móvil de grupo de investigación de Tolerancia a Fallos del Departamento de Informática de Sistemas y Computadores (DISCA) en la Universidad Politécnica de Valencia en España. Pretendiendo implementarla en un robot móvil de servicio en un futuro próximo.

1.5 Estructura del trabajo de investigación

Éste trabajo está estructurada en 7 capítulos, mismos que son descritos a continuación. En el presente capítulo se incluye la introducción y la motivación de la que surge el presente trabajo de investigación. Lo que se pretende es dar una idea general del contenido y la propuesta Arquitectura Tolerante a fallos. En el capítulo 2 se revisan las características de los agentes, los Sistemas Multiagente (SMA), y las diferentes metodologías de desarrollo que existen para la construcción de un SMA; con el objetivo de seleccionar la metodología formal más adecuada para realizar y validar el modelo del SMA propuesto en este trabajo.

En el Capitulo 3 se presenta diversas metodologías para el diseño de sistemas de control industrial tolerantes a fallos, ya que estos han sido da la base para la detección y diagnóstico de fallos en los robots móviles, asimismo se realiza el estudio de últimas formas de implementar mecanismos tolerantes a fallos en diferentes tipos los robots móviles y estáticos.

En el Capítulo 4, se efectúa el análisis de prerrequisitos para modelar la arquitectura tolerante a fallos propuesta en éste trabajo.

En el Capítulo 5 se realiza el análisis de fiabilidad de los componentes de entrada y salida que integran el sistema de control del robot, con el objetivo es diagnosticar los fallos en componentes por medio de la observación de síntomas difusos usando la información contenida en un árbol de fallos.

En el Capitulo 6 se modela la arquitectura propuesta mediante un Sistema Multiagente utilizando una metodología formal con la finalidad de que el sistema quede totalmente validado, basándonos en los requerimientos analizados en el capítulo 5.

Capitulo 7 se tratan las conclusiones y trabajos futuros.

Anexo A Caso de Meta: Tolerador de fallos a nivel Nodo en un dispositivo múltiple.
Anexo B Caso de Meta: Tolerador de fallos a nivel Nodo en un dispositivo único.
Anexo C Sheel genérico con sus tres bases de conocimiento, que se pueden integrar en los agentes que integran el Sistema Multiagente propuesto en éste trabajo de investigación.
Anexo D Tablas de las tareas y los mensajes que realizan los diferentes tipos de agentes que integran al Sistema Multiagente tolerante a fallos para el sistema de control de un robot móvil.

Capítulo 2 Sistemas Multiagente

2.1 Introducción

El hombre a lo largo de su historia ha buscado siempre mecanismos o medios para facilitar el desarrollo de sus tareas, por lo que ha pasado por diferentes eras siendo estas: era de la agricultura, era industrial, era de la electrónica. En la era industrial, por ejemplo, el hombre válido de su ingenio creó instrumentos que fueran la extensión de las capacidades típicas de su cuerpo. En la actualidad el gran cambio lo está marcando la transición del paradigma de la sociedad industrial al paradigma de la sociedad del conocimiento, por lo que se puede decir que entramos a la *era del conocimiento*, el hombre a través de la tecnología en la información busca extender las capacidades de su mente. En esta búsqueda se han definido herramientas, que tratan de expandir el conocimiento, entre ellas podemos citar a la *Inteligencia Artificia l(IA).* Dentro de éste enfoque surge como paradigma interesante *el agente software*.

Un agente según Marvin Minsky es un programa computacional con cierta inteligencia [Minsky M. 1975]; *una entidad software a la que se le puede delegar tareas.*

Un indicativo de que la tecnología de agente ha madurado dentro de una parte significativa en el área de las ciencias computacionales, es el gran número y variedad de aplicaciones que han aparecido recientemente [Jenning 96, Arens 96, Hayes 95, Ishiza 96, Lashka 94, Etzion 94]. Estas aplicaciones abarcan: interacción hombre-máquina, comercio electrónico, control industrial, robótica, acceso y recuperación de información, gestión y planificación de recursos, juegos, entornos virtuales, simulación social, interfaces inteligentes, procesos de negocios entre otras.

Como consecuencia de lo anterior, la industria ha comenzado a interesarse en adoptar esta tecnología para desarrollar sus propios productos. La introducción de la tecnología de agentes software en la aplicación de soluciones industriales, requiere de metodologías que asistan en todas las fases del ciclo de vida para desarrollar sistemas computacionales basados en agentes. Sin técnicas adecuadas para soportar este proceso, tales sistemas no son lo suficientemente confiables, sustentables o extensibles, y serán difíciles de comprender y sus elementos, no podrán ser reutilizados.

La mayor parte de problemas que se requiere resolver utilizando múltiples agentes puede ser resueltas con un sólo agente. Sin embargo, al igual que el concepto de programación descendente, modular y orientada a objetos intenta dividir los problemas en sub-problemas cada vez más sencillos [Posadas Yagë Juan L. 2003], en el ámbito de los agentes se renuncia a la idea de un agente único. Es decir, lo mismo que en la sociedad humana podemos encontrar las actividades claramente divididas para conseguir un elevado grado de especialización y eficiencia, en el caso de los agentes se tiende a éste planteamiento en el que múltiples agentes autónomos interactúan, se coordinan (cooperando, compitiendo o ambas cosas) y todo ello de un forma ádaptativa. Por lo que se constituyen sociedades de agentes a las que se les consideran Sistemas Multiagente (SMA).

Cualquier SMA ofrece importantes ventajas sobre los sistemas de centralizados, de la misma forma que la computación distribuida mejora las prestaciones de la centralizada. Parece lógico que los agentes no actúan solitariamente, sino que se localizan en entornos o plataformas con varios agentes. Cada uno de los agentes del entorno tendrá sus propios objetivos, tomará sus propias decisiones y podrá tener la capacidad de interactuar y comunicarse con el resto de agentes [Giret, Insfrán, Pastor, Cernuzzi 2000]. Los entornos o plataformas que soportan varios agentes es lo que en la literatura se conoce con el nombre de Sistema Multiagente o agencias.

El objetivo de este capítulo, es conocer acerca de la teoría de los agentes de software ofreciendo un fundamento teórico para el soporte de toma de decisiones en el momento de elegir lo que más convenga en nuestro trabajo a través del análisis del estado del arte y así poder desarrollar un Sistema Multiagente para tolerar fallos en ambientes robóticos.

Éste tema se inicia con el cuestionamiento del por qué y cuando usar un SMA en lugar de un sólo agente para solucionar un problema utilizando la computadora, y cuáles son sus ventajas y características principales, esto se describe en la sección 2.2. La base primordial de un SMA son los agentes por lo que es importante su estudio, y su modelado, así mismo se debe conocer sus características más importantes esto se presenta en la sección 2.3. La parte principal de un SMA son las interacciones entre los agentes ya que por medio de estas cooperan y resuelven un determinado problema, esto se trata en la sección 2.4. La característica fundamental para que cada agente que integra el SMA pueda actuar de manera autónoma y con razonamiento, y resuelvan el problema como lo hacemos los humanos se trata en la sección 2.5. Las diferentes técnicas de representación del conocimiento se presentan en la sección 2.6. En la sección 2.7 se presenta como modelar un SMA de manera formal, así mismo la importancia que tiene el paradigma orientado agentes en la solución de problemas, y su aplicación real, también se muestran sus aplicaciones, siendo el caso particular de este trabajo un SMA para la tolerancia a fallos en un sistema de control distribuido en un robot móvil. Por último, en la sección 2.8 se dan las conclusiones sobre este capítulo.

2.2 Cuándo usar un Sistema Multiagente

El Object Management Group (OMG) define un Sistema Multiagente como: [Giret, Insfrán, Pastor, Cernuzzi 2000] *una plataforma que puede crear, interpretar, ejecutar, transferir y terminar agentes*. Sin embargo, hay que destacar también que para poder soportar SMA es necesario disponer de un entorno adecuado para su desarrollo. Este tipo de entornos debe disponer al menos de las siguientes características:

- Contener agentes con las capacidades de autonomía, adaptabilidad, interacción coordinación.
- Partir de un planteamiento abierto donde se abandone un diseño centralizado.
- Ofrecer una infraestructura donde se pueda especificar claramente la comunicación y los protocolos de interacción entre agentes y con el propio entorno.

A pesar de éstas consideraciones y que no se cuenta con una metodología estándar para su desarrollo se están haciendo muy populares, debido a que se están realizando trabajos de investigación en todos sus ámbitos, existen SMA comerciales que se han implementado y han tenido éxito como es el caso del SMA desarrollado para control aéreo en el aeropuerto de la ciudad de Sidney en Australia [Posadas Yagë Juan L. 2003]
.

El hecho de implementar un SMA suele plantear una serie de problemas como la elección del tipo de comunicación entre los agentes, el tipo de plataforma o de la metodología de desarrollo, de los criterios para la seguridad del sistema, etc. Sin embargo se debe tener presente que un Sistema Multiagente tiene grandes ventajas con respecto a un sistema centralizado y monolítico como son:

- Solución de problemas con mayor rapidez, debido al aprovechamiento de procesamiento en paralelo.
- Comunicación mínima, pues se transmite solamente soluciones parciales de alto nivel a otros agentes en vez de tener que enviar datos básicos a un sistema central.

- Mayor flexibilidad, pues se tienen agentes con diferentes habilidades que en forma dinámica cooperan entre sí para resolver problemas.
- Mayor confiabilidad, pues otros agentes pueden tomar las responsabilidades de los agentes que llegasen a fallar en su operación.

Tomando en cuenta las ventajas anteriores el problema que se intenta resolver en éste trabajo se tiene un sin número de ventajas el modelar la arquitectura mediante un Sistema Multiagente, ya que la naturaleza del sistema de control del robot móvil es distribuida, además el problema que se requiere resolver es complejo, por lo que estamos ciertos que se toleraran aceptablemente los fallos a nivel software y hardware garantizando un sistema robótico con una mayor disponibilidad de funcionamiento, flexible, escalable con lo que se incrementa la productividad del robot.

2.3 Características de los Sistemas Multiagente

Un SMA tiene las siguientes características:

- Cada agente tiene información o capacidad incompleta para resolver el problema.
- Cada agente tiene un comportamiento concurrente. Por ejemplo, el agente puede realizar varias tareas e interactuar con varios agentes simultáneamente.
- No hay un control global del sistema.
- Los datos son descentralizados.

De acuerdo a éstas características, modelar los sistemas complejos como un SMA tiene las siguientes ventajas:

- Se evitan problemas de limitación de recursos o el riesgo de fallos de manera natural en situaciones criticas que se podrían tener al encomendar un problema complejo a un solo agente.
- Se permite la interconexión e interoperabilidad de sistemas existentes.
- Se obtiene una solución flexible a un problema en una sociedad de agentes interactuando.
- Se tienen soluciones eficientes a problemas en los que se requiere el uso de información que está espacialmente distribuida.
- Se permite la extensibilidad ya que el número y la capacidad de los agentes trabajando en el sistema puede cambiar dinámicamente.

2.3.1 Los agentes modelos y características

Los SMA pueden definirse como una red de solucionadores de problemas que trabajan juntos para dar solución a un determinado problema que está más allá de sus capacidades individuales. A estos solucionadores de problemas, se les conoce con el nombre de *agentes*, son autónomos y pueden ser homogéneos o heterogéneos en su naturaleza dependiendo de sus capacidades en la solución de problemas, en esta sección se trata muy brevemente un modelo de agente, y sus principales características.

Un agente puede ser clasificado como reactivo o proactivo.

Agente reactivo no tiene una representación interna de conocimiento y se comporta de acuerdo a estímulos o entradas que recibe del ambiente donde esta actuando.

Agente proactivo se comporta de acuerdo a un modelo de razonamiento simbólico, por medio del cual se coordina con otros agentes. Como ejemplo podemos mencionar los agentes BDI (Beliefs creencias, Desires deseos, Intentions intenciones), que se comportan de acuerdo a creencias, deseos e intenciones.

Existe otro tipo de agente siendo éste; el que muestra atributos de cooperación y aprendizaje. La cooperación entre agentes en un SMA es una interacción social por medio de un lenguaje de comunicación. Los agentes que cooperan entre si y deben conocer el mismo lenguaje [Ierache Jorge Salvador 2003].

El agente inteligente (*ideal*) tiene la capacidad para aprender a interactuar y reaccionar ante el ambiente que lo rodea.

Por último, un agente puede ser clasificado por su movilidad.

El agente móvil tiene la capacidad de moverse de un nodo a otro dentro de una red de computadoras con el objetivo de terminar su tarea. A partir de esta clasificación se pueden derivar diversos tipos de agentes. En éste trabajo se diseñan solamente agentes autónomos y estáticos que no tienen la capacidad de moverse. La figura 2.1 muestra un modelo clásico del agente y a continuación se describen sus componentes [Haddadi A.,Sundermeyer K. 1996].

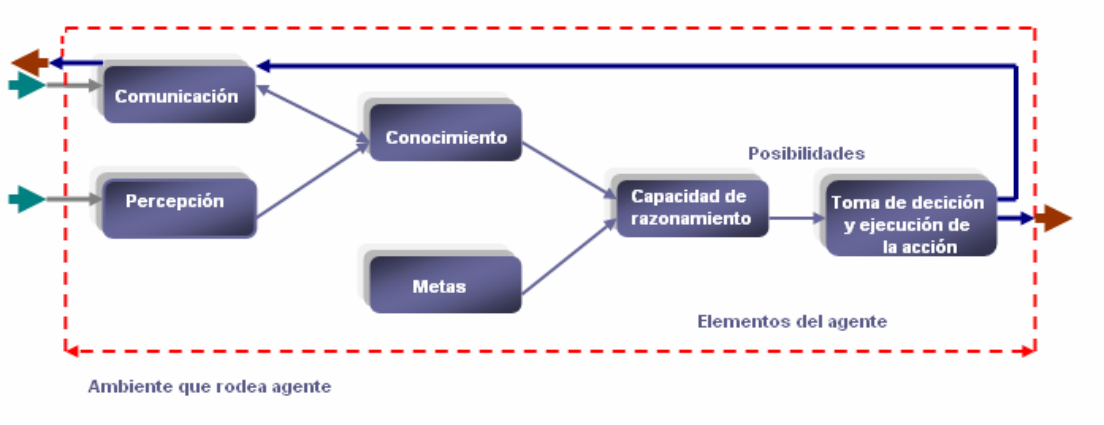


Figura 2.1 Modelo de un agente.

- Los agentes tienen conocimiento acerca del problema al que tratan de darle solución. Y pueden obtener este conocimiento cuando preguntan información al usuario, o al percibir variables del medio ambiente que los rodea, o a través de la comunicación con otros agentes.
- Tienen metas que deben alcanzar en el transcurso del tiempo.
- Son capaces de analizar un conjunto de posibles soluciones para lograr sus metas dependiendo de su capacidad de razonamiento.
- Tienen la capacidad de tomar decisiones para seleccionar la mejor opción.

El término de agente se usa para nombrar a un sistema (hardware o software) que tiene las propiedades que se muestran el la figura (2.2), las cuales sirven a su vez para definir un modelo genérico de agentes.

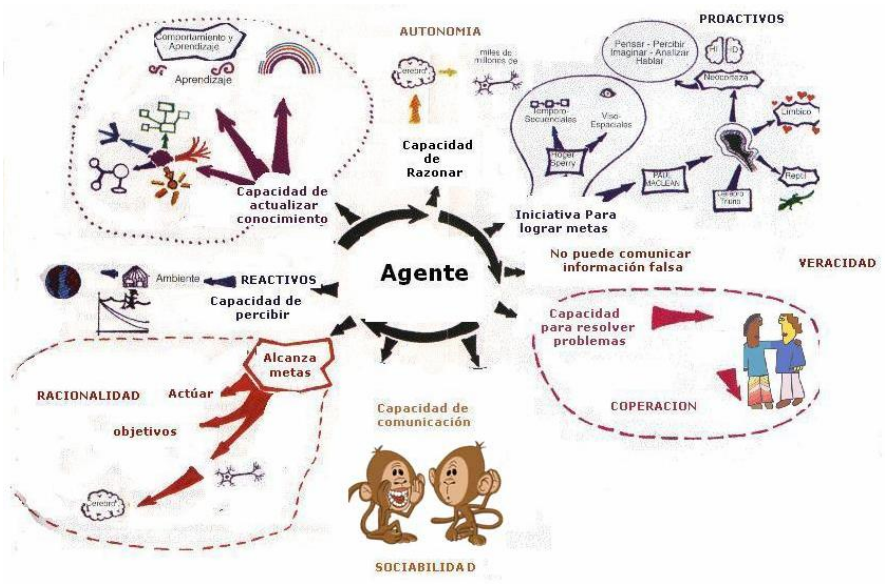


Figura 2.2 Características de los agentes.

2.4 Interacciones entre agentes en los Sistemas Multiagente

En un SMA las interacciones entre agentes son necesarias para poder lograr un objetivo común. El intercambio de información tiene la finalidad de coordinar los elementos que trabajan juntos por una meta común (coordinación), otra interacción es la sincronización de acciones antes de iniciar una actividad particular o resolver conflictos (negociación) [Posadas Yagë Juan L. 2003]. Ésta interacción se lleva a cabo intercambiando información (paso de mensajes) de modo síncrono o asíncrono y pueden ocurrir entre agentes similares o diferentes dentro del mismo ambiente o en ambientes heterogéneos. Ya que las interacciones en SMA son dinámicas, es decir, pueden crearse y terminarse en el tiempo, se han definidos protocolos de interacción y lenguajes de comunicación de agentes (Agent Communication Language ACL) para poder llevar a cabo la comunicación [Grosz B., Sidner C. 1990]. La interacción entre agentes se da por medio del intercambio de información, que puede ser de alguna de las siguientes formas:

a) Comunicación mediante mensajes que se debe establecer entre los agentes.

b) Percepción de los cambios en el medio ambiente debidos a la acción de otro(s) agente(s).

Si un agente adquiere conocimiento sobre otros agentes, puede ser capaz de reconocer las acciones y de inferir las decisiones de dichos agentes. Es posible distinguir tres tipos de intercambio entre agentes, de acuerdo al tipo de información que se intercambia: Intercambio de conocimientos, intercambio de posibles soluciones, intercambio de decisiones.

Intercambio de conocimiento: Se debe a las diferencias y a la percepción incompleta del medio ambiente que rodea a dos agentes debido al conocimiento parcial sobre la realidad. De acuerdo a lo anterior, sus conocimientos podrían ser complementarios o conflictivos acerca de una situación común. En el caso de conocimiento complementario, es útil que cada agente pueda razonar acerca del conocimiento de otros agentes para poder hacer el intercambio de información. En el caso de

conocimiento conflictivo, el problema consiste en negociar para decidir cuál de las descripciones se asemeja más a la realidad.

Intercambio de posibles soluciones: A partir del conjunto común de posibles soluciones, se requiere que los agentes se pongan de acuerdo para seleccionar una solución mutua cuando éstos se encuentran cooperando entre sí. Una opción para esta situación, consiste en ponerse de acuerdo al seleccionar una primera solución que se encuentre en la intersección de las soluciones halladas por cada agente. Sin embargo, puede ser que la solución que sea común para ambos agentes, tenga diferente calificación para cada uno. Puede ser que para un agente una solución tenga una buena calificación, mientras que para el otro, la solución común no le sea interesante. Bajo esta situación, es necesario que se establezca una negociación entre agentes [Haddadi A.,Sundermeyer K. 1996]. En el caso de interacción débil, no es necesario intercambiar planes, ni tampoco se necesitan comunicar o explicar decisiones.

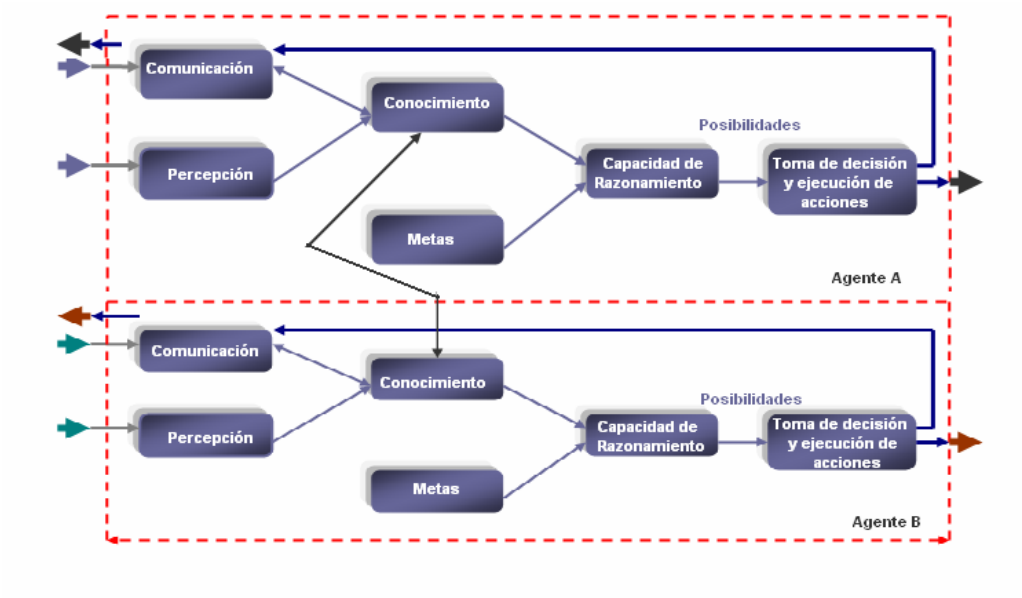


Figura 2.3 Interacción y capacidad de razonamiento débil entre agentes.

Una interacción media entre agentes sucede cuando uno de ellos toma en cuenta lo que otros intentan hacer. Un ejemplo de ésta situación se da cuando varios agentes tratan de obtener un plan común, al utilizar su propio razonamiento individual e intercambiar posibles planes hasta construir un plan donde participan todos los agentes.

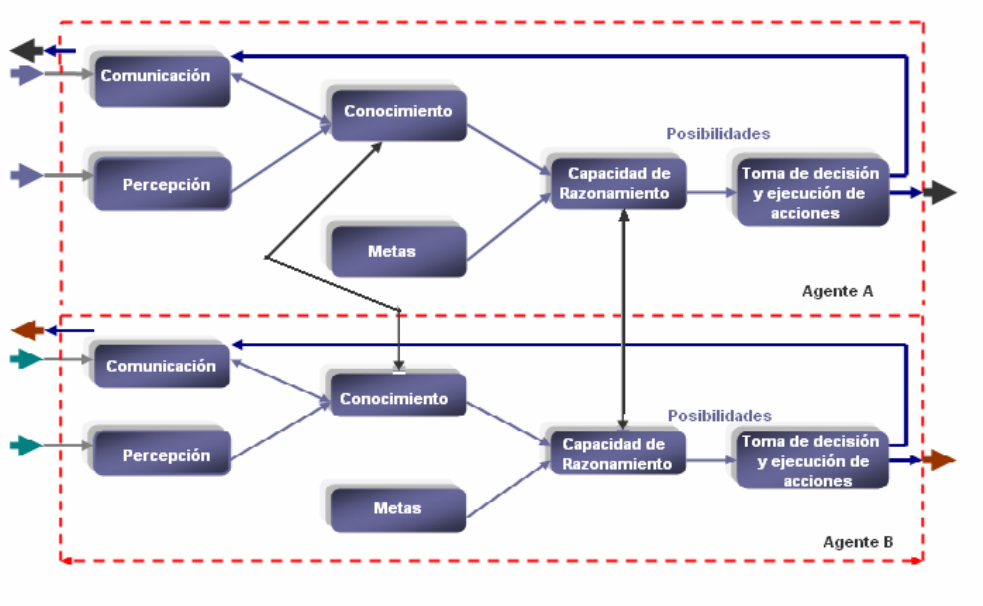


Figura 2.4 Interacción y capacidad de razonamiento medio entre agentes.

Cuando existe una fuerte interacción entre agentes, se requiere de protocolos para informar, solicitar o negociar. Por ejemplo, el solicitar no sólo está compuesto de una orden para hacer algo, sino debe ser seguido por una respuesta aceptando la solicitud, o bien por un conjunto de intercambio de mensajes para negociar y llegar a un acuerdo acerca del significado del requerimiento [Vázquez 2003].

2.4.1 Técnicas de Interacción

Una técnica de interacción entre agentes ampliamente utilizada es la comunicación directa. Una forma de éste tipo de comunicación es el paso de mensaje *punto-a-punto*. Los agentes que intervienen en la comunicación tienen que conocerse entre ellos. Éste enfoque es bastante difícil cuando no se conocen los diferentes agentes con los que se desea interactuar o el número de ellos es muy grande. Otra manera de comunicación directa es el paso de mensajes denominado comúnmente *broadcast*, en la cual, un mensaje es enviado simultáneamente a varios agentes con comportamiento similar (entienden el mismo lenguaje) siendo el más utilizado en los sistemas distribuidos. Ésta forma de comunicación tiene la desventaja de "costo de comunicación" cuando se envía información a un grupo muy numeroso de agentes o se tienen agentes con diferentes comportamientos.

Para evitar los problemas de la comunicación directa se usa la *comunicación indirecta*. En ésta forma de comunicación los agentes se organizan en grupos. Y no se comunican directamente si no por medio de un *facilitador*. Los agentes de un grupo que entienden un lenguaje pueden comunicarse con otros agentes que entienden otro lenguaje por medio del *facilitador*, pero se conserva el problema del "costo de comunicación" cuando hay numerosos grupos de agentes con diferentes lenguajes como es el caso de Internet [Haddadi A.,Sundermeyer K. 1996].

2.4.2 Colaboración, coordinación y comunicación entre agentes

Los problemas que intentan solucionar los SMA se resuelven en función del grado de cooperación o colaboración entre los agentes individuales. Es importante enfatizar en el diseño de esta faceta en los agentes, tradicionalmente se han presentado tres estrategias de interacción básicas:

- *Agentes cooperativos*: Trabajan conjuntamente con la intención de resolver juntos un problema y pueden verse como un modelo adecuado para la gestión de redes. En general, esta estrategia es útil para el control de sistemas críticos en donde se debe conocer el estado de equilibrio del sistema. Este tipo de agentes está pensado para sacrificar la utilidad individual a cambio del bien general de todo el sistema.
- *Agentes auto-interesados* son aquellos que intentan maximizar su propio beneficio sin preocuparse del interés general del sistema. En general, este tipo de coordinación es apropiado cuando los agentes van a competir en entornos abiertos.
- *Agentes hostiles*: Este tipo de agentes no tiene una aplicación directa en los agentes que integran al sistema propuesto; ya que se basan en la idea de una utilidad que se ve incrementada con su propia ganancia y con la pérdida de otros agentes competidores.

Se han realizado actualmente múltiples investigaciones en el ámbito de la planificación distribuida, la planificación general, la asignación de recursos y el control de problemas mediante agentes cooperativos. Estos sistemas ofrecen las siguientes ventajas para el trabajo que se desea realizar el presente trabajo:

- La reducción de complejidad en los algoritmos tolerantes a fallos ya que el sistema se configura ante un fallo implementando la comunicación entre los diferentes agentes que integran el sistema de manera coordinada.
- Incremento de la velocidad y optimización de cómputo a través del paralelismo.
- Incremento de la reactividad sin necesidad de consulta a entidades centralizadas.
- Robustez, disminuyendo la dependencia en sólo nodo en concreto. En algunos casos la robustez también se puede conseguir mediante la replicación de algunos agentes, en lugar de su distribución.

Los agentes que integran a un SMA deben *colaborar* entre ellos para desempeñar sus tareas, esta colaboración suele realizarse mediante un lenguaje de comunicación comprensible por todos los agentes que integran el sistema, y también por otros programas si es necesario. La diferencia entre un *agente* y un *objeto* (dentro la programación orientada a objetos) es que el último ejecuta métodos de otro objeto siempre que tenga permiso, un agente puede rechazar una petición de otro agente, por lo que deben ser capaces de comunicarse entre sí, para decidir qué acción ejecutar o qué datos obtener [Giret, Insfrán, Pastor, Cernuzzi 2000]. El mayor inconveniente para conseguir la comunicación está en el hecho que existen múltiples lenguajes propios de cada fabricante, lo que no facilita la compatibilidad en sistemas heterogéneos. Los agentes que integran un SMA deben coordinar sus actividades para alcanzar sus objetivos y ser capaces de negociar con otros agentes cuando aparecen problemas como la escasez de recursos. *La coordinación* es necesaria para determinar la organización estructural entre un grupo de agentes. La técnica *de coordinación* más exitosa para la asignación de recursos y tareas entre una sociedad de agentes es Contract-Net Protocol (CNP) adoptado en las especificaciones de Foundation for Intelligent Physical Agents (FIPA). En CNP se aplica una estructura de mercado donde los agentes toman dos roles principales, el de director y el de contratista [Hayes Roth B. 1988].

El principio básico *en la coordinación* es; si un agente no puede resolver la acción que tiene asignada usando su conocimiento local, entonces lo que requiere hacer es descomponer el problema en sub-

problemas, y buscar a otros agentes con recursos suficientes que sean capaces de solucionar los sub-problemas. La asignación de los sub-problemas se soluciona mediante un mecanismo de contratación donde un agente director crea un contrato que luego difunde a otros agentes del sistema que pueden aceptarlo o no. CNP aporta las ventajas de permitir la asignación dinámica de tareas con la participación de otros agentes.

La comunicación entre los agentes y la forma de realizarla es otra de las características importantes en los SMA. Los agentes pueden pasar información a otros agentes para compartir entre ellos los cambios producidos en el sistema o bien para informarse de los cambios previstos en el estado del sistema. Esto generalmente se realiza en los SMA en los niveles de abstracción muy altos, en lugar de usar los tradicionales métodos remotos de paso de mensajes [Giampapa J.A., Sycara K. 2003].

Normalmente, la comunicación entre agentes implica el paso de información a nivel de conocimiento, y la comunicación puede realizarse a través de mecanismos dinámicos como es la memoria compartida, los Inter Process Communication (IPC) o a través de mecanismos estáticos como es el uso de archivos con bloqueos. De este modo, la literatura nos presenta dos categorías de comunicación entre agentes; por un lado, el uso del ACL propietarios y por otro lado, el uso de ACL normalizados.

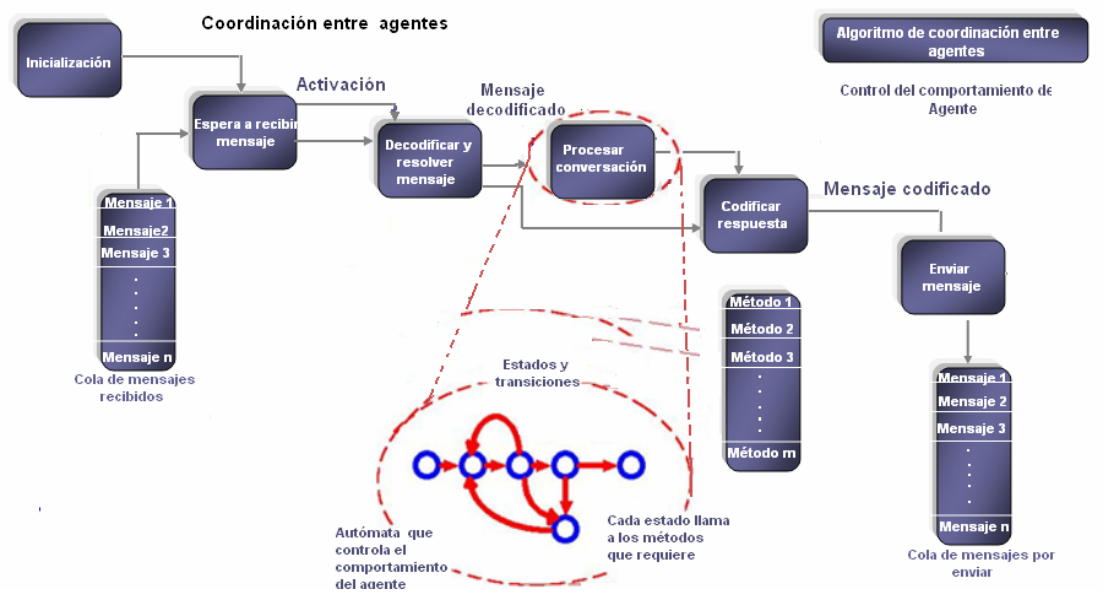


Figura 2.5 Algoritmo de coordinación de los agentes.

La figura 2.5 muestra el algoritmo general de coordinación entre agentes, y el mecanismo para controlar el comportamiento de cada agente [Vázquez 2003].

El núcleo del agente se basa en un algoritmo de coordinación, el cual se encarga de manipular el comportamiento del agente de acuerdo a los mensajes que recibe y el conocimiento que tiene.

De acuerdo a la arquitectura general de los agentes, el módulo de razonamiento puede actualizar su base de conocimientos, ya sea por los mensajes que recibe de otros agentes, o por la percepción que tiene de las variables del ambiente. El agente es capaz de tomar en cuenta los mensajes que provienen de otros agentes, o de negociar con otros agentes. Considerando al conocimiento y las metas del agente, éste módulo es capaz de evaluar diferentes alternativas de solución además de negociar y seleccionar la

mejor opción. Éste comportamiento refleja todos los estados que pueden tener las conversaciones del agente.

En la figura 2.5, se puede visualizar un primer paso que indica la inicialización del agente, quedando en estado de espera de los mensajes provenientes de otros agentes. Cuando un mensaje llega a la *cola de mensajes recibidos*, se debe decodificar el mensaje (esto lo realiza el módulo codificador/decodificador de mensajes programado en cada agente). Una vez decodificado el mensaje si no puede ser entendido por el módulo de razonamiento, entonces se envía una respuesta al agente que originó el mensaje indicando que hubo un error. Por el contrario, si el mensaje recibido lo maneja correctamente el agente receptor, entonces el módulo de razonamiento procesa la conversación. Es en éste punto el comportamiento del agente es modelado de acuerdo a una determinada conversación que tiene con otros agentes. Para responder al mensaje, se expresa en términos de estados y transiciones. En cada estado de la conversación, el agente puede hacer llamadas a métodos que le permiten hacer inferencias sobre el conocimiento que tiene para dar una respuesta, o generar un mensaje de pregunta, o solicitar alguna acción a otro agente, o utiliza la interfaz de usuario para comunicarse con él. Los mensajes de salida, son producto de cada estado de la conversación, y deben ser codificados por el módulo codificador/decodificador para posteriormente envíalos al módulo de comunicaciones *en la cola de mensajes por enviar*. Para que dos o más agentes puedan comunicarse entre sí, es necesario contar con un lenguaje de comunicación universal. Existen actualmente dos puntos de vista en el diseño del lenguaje que son: *Estilo procedural y Estilo declarativo* [Glez-Bedia M, Corchado J. M., Corchado 2002].

El estilo procedural se basa en la idea de que la comunicación puede modelarse como el intercambio de directivas procedurales. Los lenguajes de Scripts (tales como, Apple Events y Telescript, TCL) se basan en este estilo. Estos lenguajes permiten a los programas transmitir no sólo comandos individuales sino programas enteros.

El estilo declarativo se basa en la idea de que la comunicación se puede modelar como el intercambio de declaraciones (definiciones, suposiciones, etc.). Este tipo de lenguaje debe ser lo suficientemente expresivo para comunicar información de diferentes fuentes y tipos, al mismo tiempo debe ser suficientemente pequeño. Se debe asegurar que la comunicación sea posible sin tener que ser demasiado grande [Moraes Márcia Ctristina, da Rocha Costa Carlos 2003].

A principios de los años 80's, se desarrolló el Knowledge Query Management Language (KQML) como una norma para la comunicación entre agentes. Este lenguaje incluye declaraciones básicas que definen las operaciones que un agente efectúa en su comunicación con otro agente. Además, el entorno KQML se puede enriquecer con agentes especiales llamados *facilitadores* que proveen funciones adicionales.

Algunos entornos como el lenguaje Java Agent Template (JATLite) de la Universidad de Stanford han desarrollado compatibilidad con KQML. Sin embargo, han aparecido varios dialectos de KQML con una sintaxis similar, que no son completamente compatibles [Murilo Juchem 2002].

La Foundation for Intelligent Physical Agents (FIPA) propuso como norma en los inicios de los años 90's el lenguaje Arcol (desarrollado por France Telecom). Arcol conserva al igual que KQML una sintaxis similar al lenguaje LISP, pero incluye una semántica formal, lo que proporciona una base rigurosa para la interoperabilidad entre agentes y evita la proliferación de dialectos [Giret, Insfrán, Pastor, Cernuzzi 2000]. Sin embargo, Arcol carece aún de algunos aspectos deseables de un lenguaje pensado para la comunicación entre agentes, como es el permitir una mayor autonomía, una mayor heterogeneidad y el uso de dialectos abiertos. Por otro lado, Common Object Request Architecture

(CORBA) define una norma genérica para la comunicación e interacción de los agentes que integran un Sistema Multiagente creados por distintos fabricantes. La comunicación permite en un SMA el intercambio de información entre agentes para anunciar su situación actual, recursos disponibles, entorno local, etc. Mediante la comunicación de este tipo de información los agentes pueden trabajar de una forma más flexible

2.4.3 Protocolos de comunicación

Los protocolos de interacción definen la estructura de la secuencia de intercambio de información (mensajes) entre agentes. Entre los protocolos más conocidos para lograr la interacción entre los agentes se mencionan los siguientes:

Protocolos de coordinación en el cual los agentes se coordinan de tal manera que llegan a un acuerdo entre ellos, para alcanzar sus metas. *ContractNet es un protocolo de este tipo,* donde se ofrece una propuesta a los agentes y ellos deben de ofrecer soluciones a dicha propuesta eligiendo la mejor solución, se le debe informar al agente que ganó para que realice su propuesta basándose en *reglamentos* previamente definidos, mientras que los otros agentes se deslindan de la propuesta [Conry S., Meyer R.A., Lesser V. 1988].

Protocolos de negociación se utilizan cuando los agentes necesitan desempeñar una tarea con la mejor presupuesta o para resolver conflictos. Uno de los protocolos de negociación es *Workteam*. Éste protocolo permite al agente recibir la especificación de un problema. El agente puede entonces generar una propuesta la cual otros agentes pueden revisar y responder; las respuestas son revisadas y el proceso se repite.

Los protocolos de colaboración son utilizados con la finalidad de que los agentes puedan pedir ayuda a otros agentes y así resolver el problema. De tal manera, los protocolos de colaboración se utilizan para resolver problemas de gran tamaño, donde un agente individual no puede resolver [Murilo Juchem, Melo Bastos Ricardo 2002].

Existe el protocolo de colaboración donde si un tipo de agente ofrece asistencia (a otros agentes que integran el SMA) se tiene registrar dentro de un agente tipo *board* que debe ser conocido por todos los agentes. De tal manera que si un determinado agente llegase a estar ante una situación problemática puede consultar al agente *board* para obtener información de los agentes que le pueden dar asistencia. La colaboración entre agentes se da a manera de solicitud y réplica [Moraes Márcia Ctristina, da Rocha Costa Carlos 2003].

2.5 Aprendizaje y razonamiento

En virtud de que el aprendizaje en un SMA está relacionado con varias áreas de investigación, se han creado de una gran variedad de modelos de razonamiento y aprendizaje [Maes 1995]. Debido a esto se han diseñado diferentes tipos de agentes que pueden aprender en base a órdenes, a ejemplos, por observación, debido a la comunicación con otros agentes, por refuerzo, por clasificación de información, por construcción de modelos y por formas especializadas de comparación [Woolridge et al. 1994]. En la actualidad es muy frecuente desarrollar el aprendizaje y razonamiento en los agentes utilizando nuevas técnicas de Inteligencia Artificial que les permite aprender directamente de su entorno, como son: redes neuronales, razonamiento basado en casos, algoritmos genéticos y algoritmos difusos [Nwana 1996].

2.5.1 Redes neuronales artificiales

Las redes neuronales artificiales (RNA) son modelos matemáticos que intentan reproducir el funcionamiento del cerebro humano y que tiene una gran capacidad de procesar información.

Se basan en el ajuste de los pesos que se pondera en las entradas de cada neurona (elemento básico de cálculo), para calcular la salida que se requiere entregar a otras neuronas y de esta manera se obtiene un objetivo de cálculo común para toda la red. La estrategia de ajuste de los pesos constituye el aprendizaje del sistema y, junto con su topología (interconexión entre neuronas), determina su funcionalidad y prestaciones.

Una vez finalizado el entrenamiento, los pesos quedan fijos (el aprendizaje ha finalizado) y la red está lista para operar como un sistema que responde a estímulos de entrada. Existen múltiples topologías de RNA y diferentes estrategias para el aprendizaje.

Para ello se debe de entrenar conjuntos de datos suficientemente representativos de las condiciones de funcionamiento normal del proceso, y se utilizan como vectores de entrada las señales en instantes sucesivos.

Las RNA son robustas tolerantes a fallos por su diseño paralelo, flexibles y adaptables además pueden manejar datos difusos, probabilistas, con ruido e inconsistentes, Se pueden utilizar en problemas de clasificación de patrones, optimización, compresión de datos, aproximación, asociaciones, predicción etc. Por lo que, las redes neuronales artificiales son muy útiles para cualquier agente software que se diseñe para realizar cualquiera de las funciones antes mencionadas [Corchado, Pavón 2004].

Sus campos de aplicación son tan diversos como: clasificación, modelado, aproximación de funciones, reconocimiento de patrones, etc.

2.5.2 Razonamiento basado en casos

Éste es un método relativamente nuevo que encuentra soluciones en los problemas reutilizando información almacenada y conocimiento obtenido en situaciones similares previas. En esencia, permite resolver problemas nuevos mediante adaptación de soluciones que ya habían sido utilizadas con éxito en la solución de otros problemas anteriormente, un sistema basado en casos (CBR) analiza el problema, utiliza un algoritmo de indexación para recuperar casos almacenados previamente y adapta los casos recuperados para solucionar nuevas situaciones. Los CBR han sido utilizados en el diseño de SMA para identificar el asesor más adecuado de un estudiante y éste trabaje con él en su tesis doctoral. En éste sistema, cuando un agente CBR recibe datos relativos a un nuevo estudiante (un nuevo caso), se compara con datos previamente almacenados, el sistema recupera el caso más similar y devuelve el nombre del asesor más adecuado para el estudiante.

2.5.3 Algoritmos genéticos

El algoritmo genético (AG) centra su estrategia de solución en un problema basado en las características de la evolución natural, específicamente, utiliza la teoría de Charles Darwin sobre la supervivencia del más fuerte, defiende que los individuos que mejor se adaptan a la sociedad tienen una probabilidad

mayor de sobrevivir y de reproducirse, una especie completa puede hacerse más fuerte cuando sus genes más débiles se eliminen y los fuertes prevalezcan [Corchado, Pavón 2004].

Los agentes software utilizan algoritmos genéticos para crear esquemas de codificación que sean capaces de decidir si el agente es adecuado para solucionar un problema o no. Posteriormente, se crea una población de agentes codificados de modo idéntico, que se evalúa a medida que se desarrolla. Los agentes fuertes (genes) pueden reproducirse y mutar, y gradualmente el sistema global de agentes se hace más fuerte a medida que los agentes débiles terminan. Los AG proporcionan las mismas ventajas que las redes neuronales artificiales robustez y adaptabilidad, sin embargo, el proceso de adaptación suele llevar más tiempo. Los AG se utilizan cada vez más en el desarrollo de agentes, De los ejemplos más recientes en el ámbito de agentes de información y en el movimiento de imágenes [Athanázio de Cerqueira Gatti Máira, Pereida de Lucena Carlos José 07]
.

2.5.4 Lógica difusa

Es una de las disciplinas matemáticas con el mayor número de seguidores es la llamada lógica difusa, siendo la lógica que utiliza expresiones que no son ni totalmente ciertas ni completamente falsas, es decir, es la lógica aplicada a conceptos que pueden tomar un valor cualquiera de veracidad dentro de un conjunto de valores que oscilan entre dos extremos, la verdad absoluta y la falsedad total.Conviene recalcar que lo que es difuso, incierto, impreciso o vago no es lógica en sí, sino el objeto que se estudia; expresa la falta de definición del concepto al que se aplica. La lógica difusa permite tratar información imprecisa, como velocidad media, presión baja, humedad alta, en términos de conjuntos difusos que combinan reglas para definir acciones: si la temperatura es baja entonces abrir calentador. De esta manera los sistemas computacionales pueden razonar.

El aspecto fundamental de los sistemas basados en la teoría de lógica difusa [Chellas B. F. 1980] es que, a diferencia de los que se basan en la lógica clásica, tienen la capacidad de reproducir aceptablemente los modos usuales de razonamiento, considerando que la certeza de una proposición es una cuestión de grado. Formalmente se puede decir que sí la lógica es la ciencia de los principios formales y normativos del razonamiento, la lógica difusa se refiere a los principios formales del razonamiento aproximado, considerando el razonamiento preciso (lógica clásica) como caso límite. Así pues, las características más atractivas de la lógica difusa son su flexibilidad, su tolerancia con la imprecisión, su capacidad para modelar problemas no-lineales y su base en el lenguaje natural.

Algoritmos basados en la teoría de lógica difusa se pueden implementar en un SMA como interface de aprendizaje del agente almacenando conocimiento de acuerdo con la experiencia que recibe del usuario o del medio ambiente que lo rodea, pudiendo así realizar predicciones sobre futuras acciones del usuario o medio ambiente que rodea al sistema [Carbó J., Molina J. M., Dávila J. 2003].

2. 6 Modelado de un Sistema Multiagente

En un SMA general, cada agente tiene un comportamiento autónomo, ya que puede iniciar concurrentemente acciones por sí mismo y también puede tener interacciones concurrentes entre diferentes agentes del sistema. Por lo tanto, esto sugiere que modelar un SMA requiere un formalismo que exprese de manera adecuada, entre otras cosas, la ejecución concurrente de las accione internas y de las interacciones de cada agente [Brazie 1997].

2.6.1 Formalismos para el modelado de un Sistema Multiagente

Los diferentes formalismos que existen actualmente para especificar agentes deben ser capaces de representar los siguientes aspectos: [Giret, Insfrán, Pastor, Cernuzzi 2000]

- La información de los agentes acerca del ambiente que lo rodea.
- Las metas que tienen un agente por alcanzar.
- La acciones que desempeña el agente para alcanzar sus metas.
- Las interacciones de un agente con su ambiente y otros agentes.
- La actividad concurrente del agente para alcanzar su autonomía.

Por ejemplo, el agente puede recibir, procesar y enviar varios mensajes al mismo tiempo sin bloquearse (varias acciones concurrentemente). Entre los formalismos más importantes para especificar un SMA se encuentran la lógica, el álgebra de procesos, los lenguajes formales, los autómatas y las Redes de Petri [Jensen K, 1999]. Cabe mencionar que hay formalismos que se adaptan mejor que otros a ciertos tipos de agentes. Por ejemplo, la lógica es un buen formalismo para el modelado de agentes proactivos, ya que estos requieren un modelo de razonamiento simbólico para representar información que solicita el agente y comportarse de manera proactiva. Sin embargo, el pasar de una especificación lógica a una interpretación computacional concreta es complejo.

El caso particular de las Redes de Petri [Lakos Charles 1995] sobre otras herramientas de modelado es su poder de visualización y facilidad para el modelado de concurrencia, causalidad y sincronización de procesos; además tienen una base matemática para el análisis de propiedades de los modelos obtenidos. Las Redes de Petri son un buen formalismo para el modelado del comportamiento de agentes autónomos e interacciones entre ellos, ya que modelan fácilmente procesos con acciones (transiciones) concurrentes a partir de un estado (marcado) determinado, además de que implican una interpretación computacional o flujo de ejecución.

El álgebra de procesos temporizada básica, se trata de un lenguaje recursivo, secuencial no determinista. Con el álgebra de procesos se pueden modelar diferentes protocolos de comunicación entre agentes con el propósito de intercambiar información a través de un determinado medio. Tomando en cuenta que los agentes que componen el sistema se ejecutan en paralelo, en un entorno distribuido. El comportamiento global del sistema depende de los procesos y de los datos. El sistema es no-determinista. Ejemplos: Alternating bit protocol, Bounded Retransmission Protocol, Sliding Window Protocol, etc.

2.7 Aplicación de los Sistemas Multiagente

Los Sistemas Multiagente constituyen un paradigma de programación para el desarrollo de aplicaciones software. Actualmente la teoría de agentes y SMA son el centro de interés en muchas ramas de la ingeniería computacional e inteligencia artificial, y se está realizando importantes aportaciones en la resolución de problemas en diversos dominios (Medicina, Aeronáutica, Robótica, Bolsa de Valores, Comercio Electrónico, Subastas Electrónicas, Telecomunicaciones, Estrategias Militares, Control de Invernaderos, Control de los Estanques de Camarón). A continuación se exponen algunos ejemplos donde se aplica este nuevo paradigma en diferentes áreas:

Reusable Environment for Task Strutured Intelligent Network Agents (RESITINA), es un Sistema Multiagente diseñado de tal manera que sus agentes que cooperan de forma asincrónica para recuperar

información en la WEB, la finalidad de éste SMA es apoyar las decisiones de la organización, tal como en la gerencia de portafolios financieros.

Proyecto Cobas del Defense Advanced Research Projects Agency (DARPA) [Suthankar G., Sycara K. 2004]. Es un SMA compuesto por miles de agentes software de todo tipo, distribuidos en diferentes computadoras con interconexión a diferentes tipos de sensores y armas. Los agentes ejecutan tareas específicas y se comunican entre sí a través de la red de comunicaciones computacional. Sus componentes son satélites, flota marítima, helicópteros de ataque, aviones espía no tripulados, aviones de combate, infantería, tanques y vehículos de transporte contiene datos e información sobre: mapas, meteorología, videos, objetivos, rutas, armas, radares, efectivos disponibles de todas las fuentes (figura 2.6 aviones, tanques, helicópteros, naves espías, soldados, analistas, etc.). Katia Sycara es la responsable del proyecto que tuvo un costo de 50 millones de dólares.

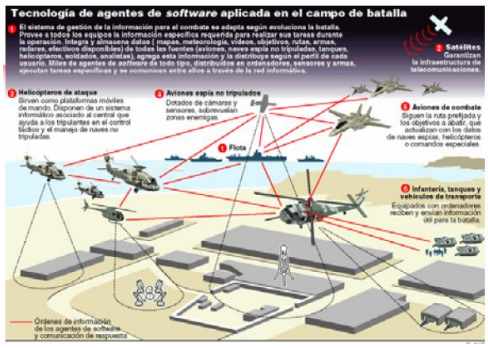


Figura 2.6 Aplicación de un SMA en el frente de batalla.

Quaky-Ant [Martínez B. Humberto 2001] es un robot móvil autónomo, su arquitectura de control es distribuida y (BGA) se basa en un SMA, de tal manera que los agentes sirven cómo base del desarrollo de los distintos elementos de control. Ésta arquitectura permite distinguir entre procesos deliberativos y reactivos. Los distintos agentes de la arquitectura se comunican por medio del protocolo KQML.

Figura 2.7 Robot Quaky Ant su arquitectura de control la integra un SMA.

OASIS es un Sistema Multiagente que controla el tráfico aéreo en grandes bloques y asigna un agente para resolver cada uno de los sub-problemas [Georgeff M. 1998]. Cada uno de los agentes resuelve parte de un nodo independiente y coopera con el resto de los agentes para definir el comportamiento global del sistema, Se diseñó un agente por cada uno de los aviones implicados, además otros agentes actúan como secuenciadores, modeladores de viento, coordinadores, controladores de trayectorias etc.

Single Point of Contact (SPOC) es un Sistema Multiagente de gestión de negocios. Este sistema se desarrolló para una gran empresa Australiana, siendo su finalidad auxiliar a los empleados de la compañía atender a los clientes. La arquitectura del Sistema Multiagente está compuesta por tres capa: Una capa del sistema está constituida por agentes que mantienen la configuración del sistema e interactúan con otros servicios. Otra capa denominada *equipo* formada por una serie de agentes que gestionan la creación y la exclusión de los agentes, y la capa del *usuario* está constituida por un

conjunto de agentes que gestionan los procesos de cada cliente al interactúan con los empleados del departamento de atención al público.

SWARMM es un Sistema Multiagente que modela un combate aéreo que permite codificar tácticas adoptadas por los pilotos en escenarios de combate y crea a los agentes con el propósito de que participen cómo pilotos reales en la simulación, el sistema modela los aspectos físicos del avión (armas, sensores, rendimiento, etc.) con el conocimiento táctico y la forma de razonar de los pilotos.

La planta ensambladora de de camiones de la General Motors en Fort Wayne, utiliza un SMA está diseñado de tal manera que los agentes realizan el control de suministro de aire para el área de pintura. Se diseñó un tipo de agente para controlar los humidificadores, otro tipo de agente para controlar los hornos, y un último tipo agente que controla los generadores de vapor para el suministro de aire. Cada agente realiza su trabajo de manera autónoma de acuerdo a las características del medio ambiente. Con la implementación de éste sistema se ha obtenido: reducción del código del software de control, adaptabilidad y flexibilidad del sistema de control, reducción en el gasto de pintura por consiguiente se ahorró dos millones de dólares anuales.

La planta de estampado de Daewoo Motors en Corea Sur ha implementó un Sistema Multiagente scheduling, de tal manera que los agentes asignan las prensas, las planchas de metal y las matrices (se estampan cinco diferentes componentes exteriores para automóvil) ante requerimientos de ensamblado de los vehículos, logrando así minimizar el tiempo de configuración de la planta.

2.7.1 Ventajas del uso de los Sistemas Multiagente

El principal interés que se tiene al pretender desarrollar un Sistema Multiagente es su robustez, la creación de programas dinámicos (instalar nuevas aplicaciones sin necesidad de paralizar el sistema, y la capacidad de que las piezas de software distribuidas en diferentes computadoras o microcontroladores se puedan comunicar entre ellos para intercambiar información a través de un lenguaje común.

Pudiendo puntualizar que la aceptación de la teoría de Agentes y Sistema Multiagente se ha incrementando con el propósito de resolver problemas complejos que anteriormente eran procesados en una sola computadora, actualmente se pueden distribuir el problema en múltiples computadoras, logrando el paralelismo de tal manera que se pueden asignar diversas tareas a diferentes agentes que componen el sistema, estos agentes trabajan cooperando para lograr una meta [Bazzan Ana 2005]
.

Los Sistemas Multiagente ofrecen escalabilidad ya que son cien por ciento modulares, de tal manera que es fácil agregar un nuevo agente al sistema y éste se adapta sin por ello requerir de ajuste alguno o rescribir todo el sistema. Por lo que diseñar un sistema de este tipo para la solución de un problema complejo conlleva mayores ventajas que desventajas.

2.8 Metodologías orientadas a modelar Sistemas Multiagente

El desarrollo de cualquier tipo de software requiere de métodos y herramientas que faciliten la obtención del producto final. En esa línea, en los últimos años han aparecido diversos trabajos de investigación que proponen procesos para el desarrollo de SMA. En esta sección se presenta los métodos más utilizados para el análisis y diseño de un SMA. Las actuales metodologías de desarrollo para SMA toman algunos de los conceptos de la Ingeniería de Software Orientada a Objetos y de la Inteligencia Artificial. Desde el punto de vista de las mencionadas tecnologías, los sistemas distribuidos

pasan tomar el nombre de SMA. Las tecnología Multiagente representan algo nuevo y excitante para el análisis, diseño y desarrollo de software complejo, a continuación se presenta la definición de lo que es una metodología para modelar software que aplica para los SMA.

Una metodología se presenta normalmente como una serie de pasos, con técnicas y notaciones asociadas a cada paso. Los pasos de la producción del software se organizan normalmente en un ciclo de vida consistente en varias fases de desarrollo [Rumbau 1991].

Ahora bien, todo los conceptos antes mencionados relacionados con los agentes y SMA, aterrizan en poder aplicarlos a un sistema que resuelva un problema determinado para ello es requisito indispensable desarrollarlo creando un modelo del SMA basado en un análisis de tal manera que su implementación sea relativamente fácil. Uno de los objetivos de este trabajo es la selección de una metodología formal de desarrollo para un SMA para lograr obtener el modelo de la arquitectura tolerante a fallos para el sistema de control de un robot móvil.

2.8.1 Diferentes tipos de metodologías utilizadas para el análisis y diseño de los Sistema Multiagente

La aplicación de la Ingeniería del Software al paradigma de agentes, es conocida como Ingeniería del Software Orientada a Agente (*Agent Oriented Software Engineering* AOSE) [Jennings, N. R., Wooldridge 1997] ha generado en los últimos años numerosos trabajos de investigación relacionados con el tema. Algunos de estos trabajos los han realizado: Michael Wooldridge, Nicholas R. Jennings, David Kinny, Carlos A. Iglesias, etc. Hasta el momento, los trabajos existentes se han centrado en intentar buscar métodos de desarrollo para poder modelar sistemas reales complejos y con características claramente distribuidas.

Estos trabajos se basan, en su mayoría, en una visión de un sistema como una organización computacional consistente en diferentes entidades interactuando. Cuando se habla de sistemas complejos, el poder identificar los diferentes subsistemas que forman parte del sistema global y las posibles interacciones y dependencias entre ellos es crucial en el momento de abordar su análisis y modelado [Oliveira, Denise,Ferreira, P. R.,Bazzan, Ana 2004]
.

Las técnicas convencionales de ingeniería *Unified Process*, CommonKADS, etc. no toman en cuenta las necesidades específicas que presentan los SMA para lograr especificar la planificación de tareas, intercambio de información con lenguajes de comunicación orientados a agentes, movilidad de código, etc. Por ello, se plantean nuevas metodologías basadas en agentes siendo:

- CoMoMAS.
- BDI.
- DESIRE.
- MAS-CommonKADS.
- MASSIVE DFKI.
- GAIA.
- AUML.
- MaSE.
- MESSAGE.
- VOWEL.
- TROPOS.

Estas metodologías parten de un modelo, informal en la mayoría de casos, de como debe ser un SMA y dan guías para su construcción. En las metodologías iniciales, las guías consisten en una breve lista de pasos a seguir. Las metodologías modernas, aunque han progresado en la integración con la ingeniería del software clásica, aún no muestran la madurez que se puede encontrar en metodologías convencionales como el *Unified Process*. El motivo principal es que sigue faltando herramientas de soporte y un lenguaje para la especificación del SMA que permitan trabajar formalmente el desarrollo de software para este tipo de sistemas.

La adopción de una orientación dirigida a los agentes en un método de desarrollo conlleva según [JenningsN.R. 2001] tener en cuenta tres elementos clave: agentes, interacciones y organizaciones. De acuerdo a esta idea, en la práctica la totalidad de las metodologías que existen tratan de dar cabida a estas abstracciones en sus propuestas.

A continuación se presenta un resumen de los distintos métodos, siendo necesario remarcar que las metodologías más recientes se alimentan en gran parte de las metodologías iniciales basadas en agentes [Giret B. Adriana 2005]. Es importante resaltar que existen dos enfoques principales que siguen las metodologías orientadas agentes siendo: la que basan en objetos, y las basadas en ingeniería del conocimiento (Knowledge Engineering KE) tratando de aprovechar las técnicas de KE para modelar características cognitivas de los agentes.

Las metodologías basadas en agentes que se extienden de las metodologías orientadas a objetos, se puede citar varias razones que justifican esta extensión del paradigma orientado a objetos y son las siguientes:

- Existen similitud entre el paradigma orientado a objetos y el paradigma orientado agentes, como establece Shoham, los agentes pueden ser considerados como objetos activos, objetos con estados mentales.
- El uso común de los lenguajes orientados a objetos para implementación de los agentes.
- La validez y prueba de las metodologías orientadas a objetos.

Las metodologías orientadas a objetos actualmente conocidas son: BDI, MaSE, AUML, DESIRE, MASSIVE DFKI, MASSAGE, VOWEL y TROPOS.

Para el caso de las metodologías de agentes extendidas de la ingeniería del conocimiento se puede citar a Glaser que indica que las metodologías de ingeniería del conocimiento pueden proveer una nueva base para el modelado de SMA dado que se ocupan del desarrollo de sistemas basados en conocimiento, tomando en cuenta que los agentes poseen características cognitivas, dichas metodologías pueden proveer técnicas de modelado de los agentes actuando cooperativamente para integrar el sistema. La definición del conocimiento de un agente puede ser considerado como un proceso de adquisición de conocimiento, sólo éste proceso se estudia en estas metodologías. Las metodologías que se extienden de de la ingeniería del conocimiento más conocidas son: CoMoMAS [Glaser 1996] y MASCommonKads [Iglesias 1998].

2.9 Características de metodologías orientadas agentes

Conceptualmente la mayoría de las metodologías orientadas agentes introducen a lo largo de su desarrollo, términos en la que se dan referencias de trabajo en la evaluación de metodologías y proporcionan información y otros puntos de vista acerca de las metodologías existentes, muy similares.

De esta forma, conceptos como objetivo, tarea, interacción, rol, responsabilidad, organización, componente y evidentemente agente aparecen en la mayoría de los trabajos de una forma u otra, y su significado suele ser bastante uniforme. La mayoría de las definiciones son muy parecidas y representan las mismas ideas [Julián Vicente, Botti Vicente 2004].

Del conjunto de conceptos que se manejan en éste paradigma se destacan: agente, rol, metas, clase, interacciones comunicación tarea, constituyendo así los aspectos fundamentales en un Sistema Multiagente. En lo que se refiere concretamente al concepto de *agente*, éste es un concepto que ya esta más que definido. Sin embargo algunos de los métodos existentes plantean definiciones propias de lo que es un agente, lo cual remarca quizás más controversia existente en éste aspecto [Julián Vicente, Botti Vicente 2004]. No obstante en la mayoría de ocasiones las definiciones únicamente varían en pequeños matices. Se puede citar algunos ejemplos: un agente es visto como una abstracción útil para resolver problemas en dominios específicos [Wood 2000], según esto, un agente vendría siendo caracterizado por lo siguiente:

- Los agentes son entidades que están distribuidas.
- Los agentes son entidades autónomas, dirigidas por objetivos.
- Los agentes comparten información con otros agentes de forma interactiva, conformando los Sistemas Multiagente.

En otro caso un agente es definido [EURESCOM 2000] a partir de un conjunto de características que deben tener siendo estas:

- Un agente tiene cierto conocimiento de su entorno.
- Un agente es responsable de alcanzar y mantener ciertos objetivos que caracterizan su conducta.
- Un agente es de ser capaz de observar el estado de su entorno y de realizar ciertos sucesos.
- Las interacciones entre agentes son descritas en términos de acciones comunicativas.
- Un agente puede ejecutar acciones que afecten a su entorno.

El concepto de *rol* es utilizado para modelar diferentes comportamientos que posteriormente un agente puede llevar a cabo. El *rol* también permite establecer cierta organización entre los agentes que componen al sistema, otorgándole ciertos privilegios a determinados roles con respecto a otros.

Otro término muy a menudo empleado es el de *interacción*. Una interacción sirve para modelar el comportamiento social de los agentes. En muchas metodologías existen modelos específicos para modelar las interacciones existiendo en la mayoría de los casos bastante similitud en el significado otorgado a éste término [Julián Vicente, Botti Vicente 2004].

Los puntos siguientes a considerar en las metodologías estarán centrados en su fase de análisis y diseño.

En lo que respecta a la fase de análisis, se pueden destacar cuatro aspectos que tratan la mayoría de las metodologías, siendo estos: la identificación de objetivos, modelado de la organización, detección de los agentes necesarios y establecimiento de las interacciones necesarias.

En lo que respecta a la fase del diseño según [Jacobson I., Christerson M., Jonson P., Overgaard G. 1999] en esta fase se trata de modelar el sistema y encontrar su forma, de tal manera que de sustento a todos los requisitos que se le suponen y que fundamentalmente son resultado de la fase previa de análisis. En el caso de los Sistemas Multiagente el diseño se centra en el modelado de la arquitectura del sistema mediante refinamiento de los modelos del análisis, el modelado de los componentes internos de

los diferentes agentes y de sus interacciones. Al igual que en la fase de análisis con respecto a estos puntos se puede resaltar lo siguiente:

El modelado de la arquitectura del sistema viene dado en la mayoría de los casos por una visión general del sistema. Esto se realiza en algunas propuestas mediante un refinamiento de la organización y de las interacciones identificadas en la fase de análisis. Este refinamiento, en general, permite el perfeccionamiento de los tipos de agentes necesarios, permitiéndose en muchos casos la adición de nuevos tipos de agentes que no hubiesen sido aún detectados o la fusión de agentes muy relacionados en uno sólo [Julián Vicente, Botti Vicente 2004].

2.9.1 VOWEL

Esta metodología fue desarrollada por el grupo MAGMA, toma en cuenta para su desarrollo principalmente los siguientes conceptos de la teoría de agentes, *entorno, agentes, interacciones y organización*. Para cada aspecto, se utilizan componentes y técnicas específicas. Para constituir agentes y esto va desde los autómatas hasta los complejos sistemas basados en conocimiento. La forma de ver las interacciones van desde modelos físicos (propagación de onda en el medio físico) hasta los actos del habla (*speech acts*). Las organizaciones van desde aquellas inspiradas en modelos biológicos hasta las gobernadas por leyes sociales basadas en modelos sociológicos [Ricordel 2001].

Figura 2.8. Arquitectura en capas en un SMA modelado con Vowel.

El propósito de la metodología *Vowel engineering* es lograr librerías de componentes que den soluciones al diseño en cada uno de estos aspectos, para que posteriormente el diseñador seleccione un determinado modelo de agente, un modelo de entorno, un modelo de interacciones y modelos de organización como se muestra en la figura 2.8. *Vowel Engineering* ha sido una de las primeras metodologías en modelar SMA utilizando diferentes vistas. El trabajo en *Vowel Engineering* está incompleto ya que no termina de estabilizarse con herramientas de soporte. Además, no existen instrucciones acerca de cómo describir a los agentes, entornos, interacciones y organizaciones [Gómez Sanz Jorge 2004].

2.9.2 GAIA

GAIA [Wooldridge 1997] es una metodología para el diseño de sistemas basados en agentes su objetivo es obtener un SMA que maximice alguna medida de calidad global.

GAIA pretende ayudar al analista a ir sistemáticamente desde los requisitos iniciales del sistema hasta el desarrollo el diseño lo suficientemente preciso como para ser implementado directamente. En esta metodología el objetivo del análisis es lograr comprender al sistema y su estructura sin referenciar ningún aspecto de la implementación. Esto se consigue a través de la idea de la *organización*. Una organización en GAIA es una colección de roles, los cuales mantienen ciertas relaciones unos con otros, sobre patrones institucionalizados de interacción. Los roles agrupan cuatro aspectos: responsabilidades del agente, los recursos que se le permite utilizar, las tareas asociadas e interacciones [Gómez Sanz Jorge 2004]. Asimismo propone trabajar al inicialmente con un análisis de alto nivel. En este análisis se usan dos modelos, *el modelo de roles* su finalidad es identificar a los roles clave en el sistema junto con sus propiedades definidas, y el *modelo de interacciones* que define las interacciones mediante una referencia a un modelo institucionalizado de intercambio de mensajes, como el FIPA-Request [FIPA 2003].
En el caso del diseño considera tres modelos: el *modelo de agentes* que define los tipos de agente que existen, cuántas instancias de cada tipo y qué roles juega cada agente, el m*odelo de servicios* que identifica los *servicios* (funciones del agente) asociados a cada rol, y el *modelo de conocidos*, que define los enlaces de comunicaciones que existen entre los agentes. Esta metodología sólo busca especificar cómo una sociedad de agentes colabora para alcanzar los objetivos del sistema, y qué se requiere de cada uno para lograr esto último [Gómez Sanz Jorge 2004].

Los principales conceptos que aparecen en la metodología se dividen en dos: abstractos y concretos. Las entidades abstractas son aquellas que son empleadas durante el análisis (figura 2.9) para la conceptualización del sistema. Las entidades concretas son empleadas en el proceso de diseño (figura 2.10). La entidad más abstracta de un sistema en la jerarquía de conceptos que se presenta es el sistema relacionado con la idea de organización ó sociedad

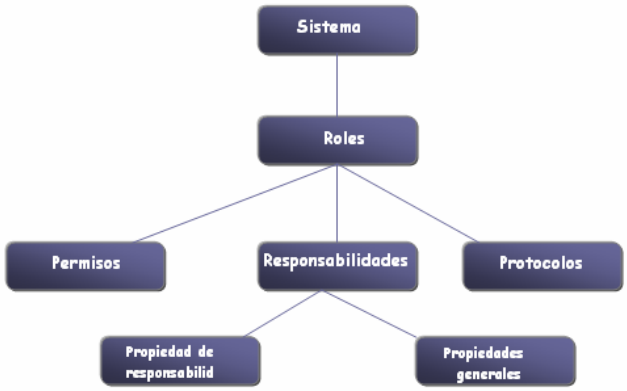


Figura 2.9 Conceptos y análisis en GAIA.

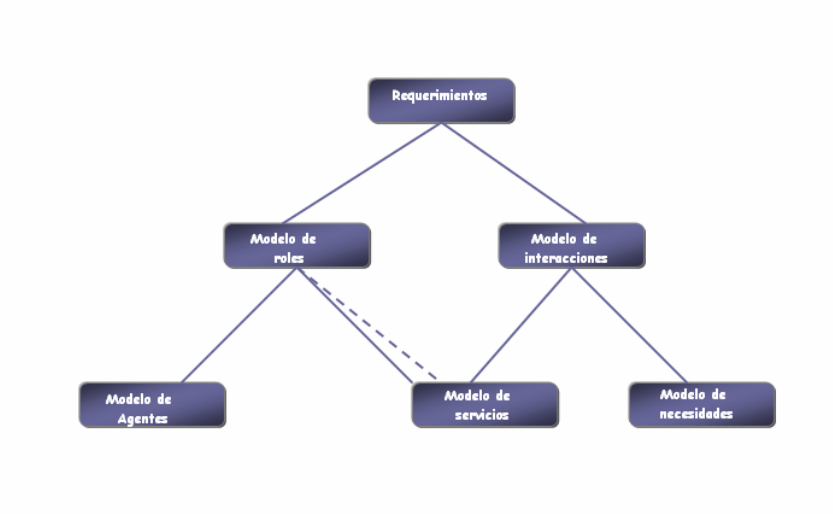


Figura 2.10 Conceptos de diseño en GAIA.

Las debilidades de GAIA [Wooldridge M. 1997], [Kinny D. 2000],[Jennings N.R., Wooldridge M. 2000] son:

1.- No se ocupa de los sistemas en los cuales los agentes no comparten metas comunes. Esta clase de sistemas representa un área de aplicación importante para los Sistemas Multiagente, y por lo tanto es esencial que una metodología deba ocuparse.
2.- Se requiere más trabajo en una estructura de organización. Estas estructuras están solamente implícitas dentro de GAIA en el rol y la interacción. Sin embargo, las representaciones explícitas de tales estructuras son de gran valor para varias aplicaciones.
3.- En lo que se refiere a la representación de los protocolos de la cooperación entre agentes es actualmente es algo pobre. Por lo que se requiere ampliar el protocolo.
4.- No fue diseñada bajo ningún estándar particular de lenguaje de comunicación entre agentes.
5.- No cuenta con una comprensión exacta de lo que significan los conceptos y los términos definidos en esta metodología

2.9.3 MAS-CommonKADS

Esta metodología se extiende CommonKADS [Iglesias F. C. Angel 2000] aplicanda ideas de las metodologías orientadas a objetos. La metodología CommonKADS gira en torno del modelo de experiencia y está pensada para desarrollar sistemas expertos que interactúan con el usuario. De hecho considera sólo dos agentes básicos: el usuario y el sistema. Este hecho influye en el modelo de comunicación que, consecuentemente, trata de interacciones hombre-máquina.

Esta metodología fue la primera en plantear la integración de un SMA con un ciclo de vida de software, concretamente el espiral dirigido por riesgos. Se plantean siete modelos para la definición del sistema: *agente, tareas, experiencia, coordinación, comunicación, organización y diseño*. Cada modelo presenta una indicación a la teoría sobre la que se basa. El modelo en sí parte de una descripción gráfica que se complementa con explicaciones en lenguaje natural de cada elemento. Existe por cada modelo una descripción de las dependencias respecto de otros modelos y de las actividades involucradas. Estos modelos se describen considerablemente en [Iglesias F. C. Angel 2000] en lenguaje natural, complementándose con otras notaciones como *Specification and Description Language* (SDL) o G. Caire *Message Sequence Chart (*MSC) para describir el comportamiento de los agentes cuando

interaccionan. MAS-CommonKADS incorpora la idea de *proceso de ingeniería* en el sentido de Pressman y describe a detalle cómo se debe definir el sistema teniendo en cuenta las dependencias entre los modelos.

Su principal desventaja es que no tiene herramientas de soporte y presenta problemas para razonar sobre la especificación de forma automática. Se realiza un gran esfuerzo en cubrir todo el ciclo de vida del sistema que se desea modelar, llegando hasta su implementación, lo cual no es frecuente. La especificación de SMA que proporciona MAS-CommonKADS detalla la mayoría de aspectos en lenguaje natural.

2.9.4 BDI

BDI (*Beliefs, Desires Intentions*) se basa en el modelo cognitivo del ser humano [Bratman 1997]. Los agentes utilizan un modelo real una representación de cómo se les muestra el entorno. El agente recibe estímulos a través de los sensores. Estos estímulos modifican el modelo del mundo que rodea al agente (representado por un conjunto de creencias). Para guiar sus acciones, el agente tiene deseos. Un deseo es un estado que el agente quiere alcanzar a través de intenciones. Éstas son acciones especiales que pueden abortarse debido a cambios en el modelo real. La concepción inicial fue realizada por M.E Bratman, fueron Georgeff, Rao y Kinny [Kinny, Georgeff 1997] quienes formalizaron este modelo y la proponen como una metodología.

Para especificar el sistema de agentes, se emplean un conjunto de modelos que operan a dos niveles de abstracción: externo e interno. Primero, desde un punto de vista externo, un sistema se modela como una jerarquía de herencia de clases de agentes, de la que los agentes individuales son instancias. Las clases de agente se caracterizan por su propósito, sus responsabilidades, los servicios que ejecutan, la información acerca del mundo que necesitan y las interacciones externas. Segundo, desde un punto de vista interno, se emplean un conjunto de modelos (modelos internos) que permiten imponer una estructura sobre el estado de información y motivación de los agentes y las estructuras de control que determinan su comportamiento (creencias, objetivos y planes en este caso).

Provee elementos para modelar y especificar Sistemas Multiagente, proponen técnicas de modelado para describir las perspectivas internas y externas de los Sistemas Multiagente basados en la arquitectura BDI. Su construcción se basa en modelos Orientados a Objetos existentes. La metodología se centra en la parte interna dejando un poco claras la forma de diseñar externamente un agente, sus interacciones y organizaciones [Gómez Sanz Jorge 2004].

2.9.5 MaSE

Multi-agent Systems Software Engineering (MaSE) [DeLoach 2001] toma las especificaciones del sistema inicial y produce un conjunto de documentos formales de diseño basado en gráficas. El principal enfoque de la metodología MaSE es ayudar al diseñador a tomar la especificación inicial del sistema basado en agentes y producir su código. Esta metodología cuenta con su herramienta de desarrollo denominada AgentTool desarrollada en *Air Force Institute of Technology* (AFIT), la cual sirve como plataforma de validación y prueba del sistema Multiagente que se desea desarrollar.

En MaSE los agentes son una abstracción conveniente, que puede o no poseer inteligencia. En este sentido, los componentes inteligentes y no inteligentes se gestionan de la misma manera dentro de una misma plantilla. El proceso para desarrollar un SMA con MaSE depende de 7 módulos (figura 2.11) donde la entada de un modulo es la salida del anterior, la mayoría de estos módulos se pueden ejecutar dentro AgentTool.

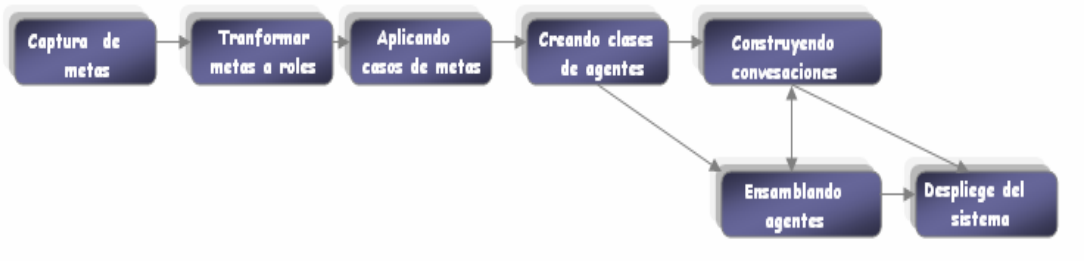


Figura 2.11 Fases de análisis y diseño en la metodología MaSE.

Su fortaleza radica en la habilidad para dar seguimiento a los cambios a través del proceso. Cada diseño se puede rastrear hacia atrás o hacia delante a través de las diferentes fases de la metodología y sus correspondientes conceptos. De esta manera, se puede dar seguimiento a la inversa para encontrar los requerimientos iniciales que apoyan a un agente en particular. Además se puede hacer lo opuesto, un determinado objeto de la fase inicial así como una determinada meta se puede mapear a un conjunto de objetos en la fase posterior. El propósito es el poder seleccionar un objeto de diseño en AgentTool y recibir una retroalimentación visual de todos los demás objetos y sus efectos.

Para el análisis se cuenta con tres pasos: capturar las metas, aplicar los casos de metas y definir los roles con sus tareas. En el caso del diseño se cuenta con cuatro pasos: crear clases de agentes, construir conversaciones, ensamblar clases de agentes y modelo del sistema. La mayoría de estos pasos se ejecutan dentro con su herramienta [DeLoach 2001].Como productos de estas etapas, MaSE espera: diagramas de secuencia para especificar interacciones, diagramas de estados para representar procesos internos en las tareas y modelar interacciones, descomposición del sistema (agente) en subsistemas (componentes del agente) e interconexión de los mismos (definición de la arquitectura del agente). Estos elementos son característicos en Unified Modeling Language (Lenguaje de Modelado Unificado UML).

La herramienta de soporte permite generar código automáticamente a partir de la especificación la generación de código, este independiente del lenguaje de programación utilizado, ya que se realiza recorriendo las estructuras de datos generadas en la especificación y generando texto como salida. No obstante, el proceso de generación de código es mejorable, ya que el código de los componentes a generar está entremezclado con el código que lee la especificación [Gómez Sanz Jorge 2004].

2.9.6 MESSAGE

*Methodology for Engineering Systems of Software Agents (*MESSAGE*)* es la una metodología desarrollada recientemente, por tanto trata de integrar resultados de las anteriores metodologías. Propone el análisis y diseño del SMA desde cinco puntos de vista para capturar sus diferentes aspectos (figura 2.12): 1.-*Organización*, que captura la estructura global del sistema. 2.- *Objetivos y Tareas*, que determina qué hace el SMA y sus agentes en términos de los objetivos que persiguen. 3.- *Agente*, que contiene una descripción detallada y extensa de cada agente y el rol que desempeñan dentro del SMA. 4.-*Dominio* en el que actúa como repositorio de información (para entidades y relaciones) concernientes al dominio del problema. 5.- *Interacción*, que trata las interacciones a distintos niveles de abstracción.

Estos elementos están presentes en los dos modelos fundamentales que propone MESSAGE: el modelo de análisis y el modelo de diseño.

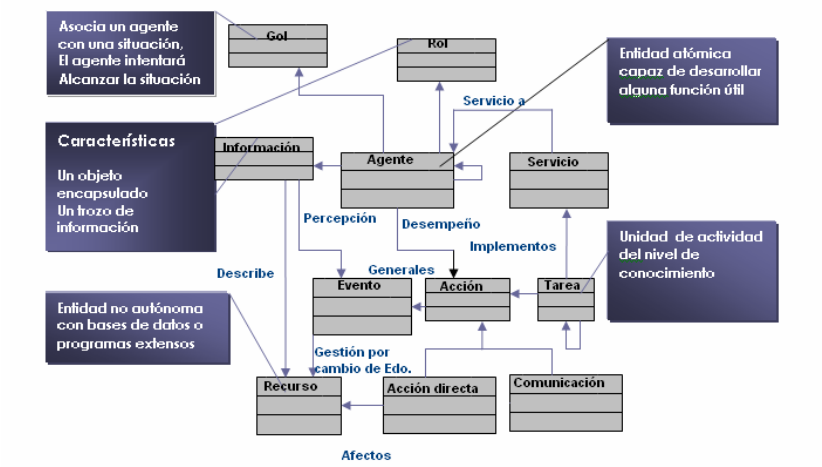


Figura 2.12 Representación de los conceptos en la metodología Message.

El modelo de análisis se limita a generar modelos a partir de meta-modelos. El modelo de diseño no llegó a concretarse completamente. Se decidió que el propósito del diseño sería producir entidades computacionales que representen al SMA descrito en el análisis. Por ello, cada artefacto producido en el análisis debería transformarse en una entidad computacional o varias cuyo comportamiento sea el que se espera en el análisis. Esto significa que las entidades del análisis se deben traducir a subsistemas, interfaces, clases, signaturas de operaciones, algoritmos, objetos, diagramas de objetos y otros.

En el modelo de diseño realmente no presenta un único proceso sino que plantea dos posibles aproximaciones:

1.- Aproximación el diseño es dirigido por la organización del Sistema Multiagente y la arquitectura.
2.- Aproximación está orientada por una plataforma de agente concreta.

MESSAGE aporta mejoras en cuanto a conceptos de ingeniería respecto de las alternativas existentes, entre ellas el desarrollo dentro de un paradigma de ingeniería del software (el Proceso Racional Unificado), aportación de métodos para la traducción de entidades de análisis a entidades de diseño y guías para la generación de los modelos. Sin embargo, los objetivos de MESSAGE no se completaron totalmente [Gómez Sanz Jorge 2004].

A favor de MESSAGE hay que destacar que ha sido la primera metodología en utilizar una herramienta para soporte del proceso de especificación de SMA de forma visual, como en UML [OMG 2001]. En cuanto a la implementación, MESSAGE provee guías de posibles arquitecturas y componentes a utilizar en esta etapa. Basándose en estas guías y los modelos de análisis y diseño, se realizó manualmente la implementación, lo cual hizo que se detectaran incorrecciones en las definiciones iniciales de los modelos [G. Caire, F. Garijo, J. Gomez, J. Pavon, E. Vargas 2000].

.

2.9.7 TROPOS

La metodología Tropos es un proyecto académico, el cual adopta un marco de trabajo ya existente denominado i*. Dicho marco propone el uso de conceptos tales como actor, dependencia social, objetivos, objetivos abstractos, tareas y recursos. Esto hace que Tropos sea una metodología orientada a los requerimientos, algo que las demás metodologías no toman mucho en cuenta.

La metodología abarca desde la etapa de análisis de requisitos hasta la de implementación. En cada una de estas etapas se utilizan los conceptos propios de *i** (actor, dependencia, etc.) para mostrar una vista del modelo teniendo en cuenta la etapa. En las vistas correspondientes a las etapas de análisis de requisitos, los actores se utilizan para modelar los *stakeholders* del dominio y el nuevo sistema a construir. De esta manera, las dependencias representadas en las vistas serán dependencias entre *stakeholders*, entre éstos y el sistema software [Gómez Sanz Jorge 2004].

La etapa de diseño, los actores modelan los componentes de la arquitectura del sistema y los agentes software que finalmente se implementaran. Las dependencias, representan pues, el intercambio de datos y control entre componentes y agentes, y definen las habilidades o responsabilidades asignadas a cada uno y que deberán ser implementadas.

Beneficios

- Pone mucho énfasis en la captura de requerimientos.
- Define un paso previo a la construcción del sistema, que tiene que ver con entender el ambiente en donde va a existir dicho sistema (entender los aspectos del negocio).
- Utiliza patrones de diseño tanto para el esquema básico del sistema como para la
- arquitectura propia de cada agente (esto permite manejar requerimientos no
- funcionales).

Problemas:

- Alto costo de aprendizaje debido a la utilización de patrones de diseño.
- Cada etapa de la metodología abarca demasiado trabajo.
- No se explica tan detalladamente la transición de una etapa a la siguiente.
- No existe una consistencia bien definida entre los modelos de i*.
- El nexo entre el diseño y la implementación que se explican en TROPOS, hacen referencia a una herramienta específica, JACK.

2.9.8 AUML

La metodología *Agent-based Unified Modeling Language* (AULM) desarrollada por Paranuk y Odell, Describe el desarrollo de Sistemas Multiagente desde el análisis al diseño, es decir, cubre algunas de las etapas el ciclo de desarrollo. En particular, se enfoca en la representación de agentes y los protocolos de comunicación. La FIPA y la OMG formaron un grupo de trabajo para agentes ellos están estudiando y recomiendan usar la metodología AUML [Odell J., Parunak H., Bauer B. 2000] . *Agent Interaction Protocol(*AIP): es el lenguaje común para comunicarse entre los agentes, la comunicación es se realiza por medio de secuencia de mensajes. En AUML se desarrollan paquetes, ya que en estos se pueden hacer plantillas y adicionalmente dentro de los paquetes puede haber procedimientos y protocolos, esta metodología toma los siguientes puntos para su operación:

- Se centra en intentar emplear herramientas de desarrollo ya existentes, como puede ser el caso de UML
- Orientándolas hacia el campo de los agentes. La visión que se presenta de un agente es como el siguiente paso a partir del concepto de objeto
- AUML sintetiza el interés por disponer de metodologías de desarrollo orientadas a agentes con la aceptación de UML.
- UML es insuficiente para modelar agentes y sistemas basados en agentes.

Si se comparan los objetos (de la programación orientada a objetos) con los agentes estos son activos. Sus actividades incluyen objetivos y condiciones que guían la ejecución de las tareas definidas, así mismo toman la responsabilidad de sus necesidades. Los agentes actúan de igual forma solos o con otros agentes, por lo que forman una comunidad social de miembros interdependientes que actúan de forma autónoma. UML en la actualidad es aceptado y es evidente que alguna de sus herramientas se puede aplicar directamente a sistemas basados en agentes adoptando algunas convenciones. Hoy por hoy se está trabajando en esta aproximación, sugiriéndose extensiones a UML para que se de soporte a la funcionalidad adicional que aportan los agentes [Gómez Sanz Jorge 2004].

2.9.9 INGENIAS

INGENIAS Se desarrolló a partir de la metodología MESSAGE Por el grupo GRACIA de la Universidad Complutense de Madrid, siendo la tesis doctoral de Jorge J. Gómez Sanz y sus asesores Juan Pavón y Francisco Garijo

INGENIAS se desarrolla a partir de la metodología MESSAGE, y como esta define un conjunto de meta-modelos (una descripción de alto nivel de qué elementos tiene un modelo) con los que hay que describir el sistema. Los meta-modelos indican qué hace falta para describir: agentes aislados, organizaciones de agentes, el entorno, interacciones entre agentes o roles, tareas y objetivos. Estos meta-modelos se construyen mediante un lenguaje de meta-modelado, el GOPRR (*Graph, Object,Property, Relationship, and Role*) [Lyytinen et al. 99].

Un meta-modelo define las primitivas y las propiedades sintácticas y semánticas de un modelo. A diferencia de otros enfoques más formales, como Z , los *meta-modelos* están orientados a la generación de representaciones visuales de aspectos concretos del sistema de forma incremental y flexible. Los *modelos* crecen incorporando más detalle debido a que no es necesario que se instancien definitivamente todos los elementos del *meta-modelo* para tener un modelo. Como se ha manifestado UML [OMG 00], construido también con *meta-modelos*, este tipo de notación facilita enormemente el desarrollo de sistemas. Otra ventaja de utilizar *meta-modelos* es que las especificaciones generadas de SMA son lo suficientemente estructuradas para ser procesadas de forma automática. Así, se puede plantear la verificación automática de la construcción de los modelos para ver si cumplen restricciones identificadas por el diseñador, generar documentación del sistema en diferentes formatos de diferentes partes de los modelos, e incluso establecer la generación automática de código desde los requisitos del SMA en los *modelos* [Gómez Sanz Jorge 2004].

INGENIAS mejora MESSAGE en tres aspectos:

- Integración de las vistas de diseño del sistema.
- Integración de resultados de Investigación.
- Integración del ciclo de vida de desarrollo de software.

En INGENIAS se toma la definición de Newell [Newell 82], una definición de agente influenciada por la IA. Newell asume que existen diferentes niveles de abstracción en el diseño de sistemas, entre ellos el *nivel de símbolos* y el *nivel de conocimiento*. Del mismo modo que un programa convencional maneja símbolos (expresiones, variables) en el *nivel de símbolos*, un agente, que equivale a un programa en el *nivel del conocimiento*, maneja únicamente conocimiento y se comporta de acuerdo con el *principio de racionalidad*. Este principio estipula que un agente emprende acciones porque está buscando satisfacer un objetivo.

Se ha tomado esta definición porque establece que el concepto de agente es simplemente una forma de ver un sistema, como también indica Shoham [Shoham 1993], que luego puede reflejarse *a nivel de símbolos*, con un programa tradicional. Esta concepción es compatible con el ciclo de vida del software. De forma general, se pensaría en el agente como un ente que se comporta de acuerdo con el *principio de racionalidad*. Cuando se requiere desarrollar físicamente un agente, se debe de poner énfasis a su arquitectura y componentes, con lo que está dentro del *nivel simbólico* [Gómez Sanz Jorge 2004].

Ventaja de INGENIAS sobre otras metodologías que está siendo probada con SMA cercanos a la complejidad de un desarrollo industrial [Gómez Sanz Jorge 2004].

2.10 Conclusión del capítulo

La necesidad de construir aplicaciones complejas compuestas de multitud de subsistemas que interaccionan entre sí es el marco de la distribución, inteligencia y de múltiple agentes. En este tipo de sistemas, donde la utilización de agentes y sus técnicas nos permite gestionar de manera inteligente un problema complejo, coordinando los distintos subsistemas que lo componen e integrando objetivos particulares a cada subsistema en un objetivo común. Este tipo de sistemas se están empleando en problemas físicamente distribuidos, cuando la complejidad de la solución requiere de experiencia muy heterogénea (involucra problemas muy distintos), o cuando la complejidad del problema es tal que el sistema debe adaptarse a cambios en la estructura o en el entorno. Considerando cada subsistema con una capacidad de decisión local, el problema de gestión se puede abordar desde una perspectiva de cooperación coordinada y negociada entre los agentes. En este caso, el problema se puede plantear como un objetivo que no se puede alcanzar por un único subsistema y necesita de la colaboración de los demás para obtener la solución. Por consiguiente, en el mundo de los Sistemas Multiagente, también conocido como la agencia, actualmente existen infinidad de campos de aplicación en los que se están obteniendo soluciones prometedoras, que los expertos coinciden en señalar cómo grandes áreas de desarrollo para ser diseñados en función de agentes software.

El concepto de agente como el de objeto, han irrumpido con fuerza en el mundo empresarial, comercial, control industrial, robótica, militar, financiero, agrícola, médico educativo, etc. y se están desarrollando multitud de aplicaciones que hacen uso de este nuevo concepto. Nos hemos acercado a los desarrollos que se está plateando, a la importancia de los mismos y a la evolución que se supone tendrán estos agentes en el futuro.

El paradigma orientado a agentes fue seleccionado ya que el sistema a desarrollar es complejo y distribuido, ideal para proponer agentes que se ejecutan autónomamente con requerimientos de cooperación coordinadamente todos entre sí para lograr el objetivo general. En el caso particular que nos atañe el objetivo es la tolerancia a fallos a nivel hardware y software del sistema de control para un robot móvil. Concretamente, nos centramos en un problema puntual donde los agentes se deben de encargar de solucionar una situación que las propuestas estándares de tolerancia a fallos no resuelven como son las pérdidas del control en determinados lapsos de tiempo de tiempo cuando se aplican algoritmos como la vuelta atrás entre otros. Con este tipo de sistema se pretende tolerar los fallos a nivel hardware y software de una manera menos complicada, debido a que se debe de dividir el problema en

piezas de software inteligente que trabajan cooperativamente para tolerar los fallos a nivel, dispositivos de entrada y salida, a nivel de las tareas, a nivel del microcontrolador, a nivel controlador de la red.

El sistema computacional que se propone es complejo, ya que se requiere manejar un gran número de variables de manera puntual, este problema se debe de resolver relativamente fácil tomando en cuenta los conceptos teóricos antes expuestos si se realiza un buen análisis y diseño, bajo la guía de una metodología de desarrollo formal orientada agentes donde estamos completamente seguros que la transformación a su implementación debe ser relativamente fácil y sin la presencia de ambigüedades, ya que si seleccionamos la metodología formal correcta que tenga una herramienta para realizar el modelo que lo valide.

Partiendo de la definición de agente de Newell, se ha hecho un breve recorrido por la teoría de los agentes, sus diferentes arquitecturas, tipos, función, características, lenguajes y plataformas de desarrollo, de la misma manera se ha estudiado las diferentes definiciones de SMA, y metodologías de desarrollo para su implementación. Durante dicho recorrido, se ha mostrado la necesidad de aplicar metodologías para estructurar el desarrollo de SMA. Saber diseñar agentes y encajarlos en un sistema no es suficiente. Un sistema debe satisfacer las necesidades del usuario que lo solicitó. Es importante saber tomar decisiones fundamentales para poder elegir las cualidades que se requiere que presenten los agentes (utilizando las diferentes arquitecturas), decidir qué entidades del sistema van a ser agentes (utilizando la definición de Newell) y organizarlo todo según una plantilla de software o utilizar una plataforma de desarrollo. Debido a que éste proceso no es trivial, se requiere del soporte de una metodología que guíe al ingeniero a lo largo del análisis y diseño del SMA que se desea desarrollar. Por tal motivo la última parte del capítulo se ha dedicado describir los diferentes tipos de metodologías que existen para trabajar con agentes, con la finalidad de seleccionar la más adecuada para desarrollar nuestro SMA.

Elegir una metodología que se ajuste a nuestros requisitos no es nada fácil, sin embargo al realizar el estudio anterior y teniendo claro lo que se quiere desarrollar, es posible seleccionar la metodología que este de acuerdo a nuestras necesidades, sabiendo de antemano cada una de las metodologías existentes en la actualidad tienen sus ventajas y limitaciones, el punto es encontrar el equilibrio. En lo que se refiere a nuestro caso particular nos decidimos por la metodología MaSE tomando en cuenta las siguientes características:

- Cuenta con herramienta de desarrollo para trabajar con la mayoría de los pasos de los que consta la metodología.
- Su arquitectura está soportada matemáticamente.
- Los requisitos del sistema los convierte en metas que casi nunca cambian durante el desarrollo del sistema.
- Trata todo el ciclo de desarrollo desde la descripción del problema hasta la implementación del SMA a desarrollar.
- Varios simuladores de SMA con elevado grado dificultad se han desarrollado utilizando MaSE.

Capítulo 3 Tolerancia a fallos en robot móviles

3.1 Introducción

Un robot es un sistema complejo integrado por dispositivos mecánico, electrónico y un conjunto de subsistemas como son: el sensorial (con el cual interactúa con su entorno), el de manipulación, el de software para ejecutar tareas de planeación, navegación, etc. La gran cantidad de componentes de hardware y software que lo integran sirven para su correcto funcionamiento. Cada uno de estos componentes que integran al robot deben ser lo suficientemente confiabilidad. La confiabilidad es particularmente importante en todo sistema robótico para realizar sus objetivos y lograr mayor tiempo de disponibilidad.

En los últimos años los diseñadores de robots móviles están incorporando en su diseño algunos mecanismos tolerantes a fallos, ya que los robots deben de realizar tareas en ambientes en la que a los humanos nos resulta peligroso trabajar, siendo el caso de: áreas radioactivas, fondo de los océanos, misiones espaciales, desactivación bombas explosivas, rescate de vidas humanas en áreas conflictivas, exploración de volcanes, trabajos de soldadura en líneas de manufactura, ayudantes de cirujanos en instalación caderas artificiales, en operaciones quirúrgicas de corazón abierto, transplantes de órganos donde se requiere alta precisión. Un caso muy peculiar es la telecirugía que emplea robots controlados de forma remota por cirujanos expertos, en estos robot se está realizando investigaciones con la finalidad de que efectúen operaciones médicas en los campos de batalla distantes a los hospitales ubicados en ciudades como Houston, Nueva York., Londres, Berlín, Paris Madrid, El Cairo, etc.

Por esta razón los fallos en el sistema de control de un determinado robot que realizan este tipo de trabajos pueden causar pérdidas humanas o tener altos costos monetarios. Para evitar fallos en los robots, y que estos no causen serios problemas, es conveniente diseñarlos con estrategias tolerantes a fallos.

Un ejemplo, es la sonda espacial no tripulada *Galileo* (Figura 3.1), diseñada por ingenieros de la National Aeronautics and Space Administration (NASA), la cuál viajó a Júpiter en 1996 para realizar tareas de detección de contenido químico de la atmósfera joviana y exploró sus lunas, contaba con una antena de comunicación desplegable, la cual sufrió un fallo y no se desplegó completamente. Tras varios intentos para lograr su despliegue, los científicos de la NASA decidieron emplear la estrategia de compresión del software, que resolvió el problema de transmisión, la sonda tenía algunos subsistemas redundantes sin embargo esta demostró que no es la única alternativa, Los ingenieros tenían a su favor que la sonda viajaría durante años para alcanzar su destino, y que la podían reprogramar, sin embargo existen casos de operaciones en tiempo real u operaciones totalmente autónomas en donde las estrategias tolerantes a fallos, son de vital importancia.

Figura 3.1 Sonda espacial Galileo.

La capacidad de detectar y de tolerar fallos en el software y hardware debe permitir a los robots hacer frente con eficacia a los fallos con el propósito de que continúen realizando las tareas para lo cual fueron diseñados sin la necesidad de intervención humana inmediata.

Tomando en cuenta que la mayoría de los robots estudiados no tienen implementado de forma integral mecanismos tolerantes a fallos, nos tomamos a la tarea de propone el diseño de una arquitectura modular, flexible y escalable tolerante a fallos a nivel software y hardware, que es posible sobreponerse en casi cualquier sistema de control de un robot móvil, con el objetivo de obtener mayor fiabilidad y disponibilidad.

Por lo que se puede decir que la tolerancia a fallos es la utilización de técnicas que aseguran que los fallos en un sistema no conduzcan a errores o a caídas. Una caída en el sistema es un evento que ocurre en cierto momento el cual provoca que el sistema no proporcione el servicio esperado.

La habilidad de seguir proporcionando un servicio considerado satisfactorio con los recursos funcionales restantes se le conoce como *tolerancia a fallos* con degradación paulatina. La degradación puede ser en desempeño, en funcionalidad, o en ambas. El contar con un sistema tolerante a fallos significa mejorar la confiabilidad del sistema [Félix Ingrand 2003].

Los fallos en un robot son fácilmente propagados por todo el sistema. Un ejemplo, el regulador de los servo actuadores que son la parte más crítica del sistema; una vez que un determinado fallo ocurre en el regulador del actuador, es muy probable causar un fallo fatal instantáneamente en todo el sistema. Generalmente, los actuadores que controlan al robot en el procesamiento de la información deben ser rápidos. En los robots industriales, el control del muestreo al azar está frecuentemente en el orden de milisegundo o a veces menor a esto. Para responder a tales procesos rápidos, los métodos de diagnóstico en muchos robots industriales se basan en limitadores del hardware, o limitadores del software o tapones mecánicos. Estos limitadores y tapones están para tomar los datos sobre cierto límite y así pueden funcionar únicamente en el área limitada, el método no puede cubrir toda la gama del movimiento del robot.

Los robots móviles deben de contar con un comportamiento de supervivencia para que en realidad sean autónomos, para poder aprovechar mejor sus recursos y obtener mejores resultados en la realización de los trabajos para lo cual se diseñaron.

Nada es perfecto y los diseñadores de éste tipo de sistemas sofisticados deben de tomar en cuenta la ley de Murphy *si algo puede salir mal, saldrá mal*.

Es posible minimizar la ley anterior, al implementar en los robots mecanismos tolerantes a fallos, de tal manera que se pueda identificar el fallo, aislarlo, recuperarse y reconfigurar al sistema para que continúe trabajando, a medida de sus posibilidades.

EL capítulo se inicia con la aplicación de los robots, y la necesidad de implementar en su diseño mecanismos tolerantes a fallos para lograr mayor fiabilidad requerida debido al trabajo que realizan, ya que un fallo en un determinado momento podría causar perdida de vidas humanas y grandes costos económicos. En la sección 3.2 Se presentan los antecedentes de los sistemas tolerantes a fallos en los sistemas de control y en los procesos industriales, debido a que los existentes mecanismos tolerantes a fallos que se han implementado en robots móviles, se derivan de éste tipo de sistemas de control. En la En la sección 3.3 Se describe cómo se ha realizado los diagnóstico de fallos en algunos robots móviles. Dentro de la sección 3.4 se detallan las técnicas tolerantes a fallos implementadas en el diseño de dos robots móviles, y en un robot estático. Por último, en la sección 3.5 se dan las conclusiones sobre éste capítulo.

3.2 Antecedentes

El concepto de control tolerante [Zhang Y. 2003] se conoce cuando se implementan mecanismos de tolerancia a fallos en la industria aeronáutica, y se trata de la implementación de técnicas tolerantes a fallos cuando se realiza el diseño de un sistema automático de control siendo estos muy propensos a producir fallos en sus componentes (sensores, actuadores, microcontroladores) durante su operación.

Los sistemas (aviones, trenes, automóviles, etc.) y procesos tecnológicos (redes energía, agua, etc.) se han automatizado y aumentado su complejidad, al incrementar el número de variables y parámetros que son medidos, así mismo se ha incrementado el número de actuadores que se accionan automáticamente en tiempo real, y por consecuencia aumenta la probabilidad de que se produzca un fallo, estos sistema debido a la función que realizan requieren de mayor disponibilidad y de un correcto funcionamiento, siendo actualmente una prioridad para los diseñadores de éste tipo de sistemas. La posible aparición de fallos puede causar riesgo para los trabajadores, inconvenientes para los usuarios y pérdidas económicas para los dueños de las industrias.

En la actualidad se diseñan sistemas de control industrial con la capacidad de responder a fallos inesperados ya sea en el hardware o software, implementándoles cada vez mejores mecanismos tolerantes a fallos.

La forma en que distintos autores enfocan el problema del tratamiento de fallos (detección, diagnosis y recuperación) es muy variada, debido en gran medida a la diversidad de tipos de fallos existentes y los métodos de recuperación disponibles. Esta disparidad se hace también patente en la terminología, dando lugar a que la nomenclatura utilizada por los diversos autores que no siempre coincide. Para unificar criterios en cuanto a definiciones de los términos empleados en éste tema, el comité técnico *International Federation of Automatic Control (*IFAC*)* denominado *(Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS*) [Iserman 1997], ha promovido una iniciativa para definir una terminología común que sirve como referencia a lo largo de éste trabajo. A continuación se describen algunos de estos términos.

En lo referente a estados y señales, se utilizan los siguientes conceptos:

Fallo Desviación no permitida de al menos una característica, propiedad o parámetro del sistema con respecto a su valor aceptable, nominal o estándar.
Avería Interrupción permanente de la disponibilidad del sistema para realizar una función requerida bajo condiciones de operación especificadas.
Error Desviación entre el valor medido o calculado a partir de una variable de salida del sistema y el valor correcto, de acuerdo con las especificaciones o cálculos teóricamente.

Residuo Indicador de fallo, basado en una desviación entre medidas del sistema y cálculos obtenidos del modelo del mismo.
Síntoma Desviación de un parámetro observable de su valor nominal (valor o rango de valores que puede tener cuando el sistema funciona correctamente).

En lo que respecta a propiedades del sistema, se distinguen los siguientes conceptos:

Fiabilidad Disponibilidad del sistema a realizar una función requerida bajo ciertas condiciones, en un determinado ámbito y durante un período de tiempo establecido.
Seguridad Disponibilidad de un sistema a no causar daños a personas, equipos o al entorno.
Disponibilidad Probabilidad de que un sistema opere satisfactoriamente en cualquier momento arbitrario.
Monitorización Tarea continúa en tiempo real que determina las condiciones del sistema recopilando información del mismo, reconociendo y notificando anomalías.
Supervisión Además de la monitorización del sistema, toma las acciones oportunas para mantener la operación en caso de fallo [Ors C. R, Serrano J. 2000].

Etapas requeridas para tolerar un fallo:

Detección del fallo Determinación de la existencia de algún fallo en el sistema, la mayoría de los fallos ocurren en errores lógicos, contándose con técnicas de detección para los mismos, como lo son: paridad y revisiones consistentes. Que ocurren en un período arbitrario en el cual no se detecta el fallo y que se conoce con el nombre de *tiempo latente o fallo latente*.
Ubicación del fallo Se trata de aislar el fallo, determinar la posición del mismo, tanto en lugar como tiempo, es necesario acotar el alcance de sus efectos a un área específica del sistema para evitar la posible propagación en otras áreas. El acotamiento de un fallo puede lograrse mediante el uso de elementos de detección y revisiones consistentes.
Identificación del fallo Después de la detección y ubicación, se estima el tipo, naturaleza del fallo y dimensiones o alcance del mismo.
Diagnosis del fallo Incluye los tres anteriores. Es decir, determina la existencia del fallo, posición, tipo y tamaño
Enmascaramiento de fallo Las técnicas de enmascaramiento del fallo ocultan los efectos de éste, sin proveer detección del mismo; sin embargo, la mayoría de las técnicas de enmascaramiento del fallo pueden extenderse para proveer detección del fallo. La técnica *mayoría de votos* es un ejemplo de enmascaramiento del fallo
Reintento de procesamiento En muchas ocasiones un nuevo intento de procesamiento puede ser satisfactorio siendo muy común en fallos transitorios [Ors Carot R, Serrano J. 2000].

Los sistemas y procesos tecnológicos complejos al automatizarse mediante lazos de control, siguen siendo susceptibles a fallos y se pueden amplificar debido a que el lazo de control podría provocar un mal funcionamiento. Además, los lazos de control pueden ocultar los fallos evitando ser observados hasta alcanzar un grado tal que producen una avería irreparable que obliga a detener el sistema o proceso. Por esta razón existe un gran interés en desarrollar sistemas de control que alcancen operar después de la aparición de un fallo y que sean capaces de detener el proceso antes de que se originen daños irreparables en el mismo [Puig, V., Quevedo, J., Escobet, T., Morcego, B.; Ocampo, C. 2004].

Existen dos enfoques para resolver el problema de control tolerante:

El control tolerante Pasivo utiliza la propiedad que tienen los sistemas realimentados de hacerle frente a las perturbaciones, cambios en la dinámica del sistema e incluso fallos. Un cambio inesperado en el sistema crea un efecto sobre el mismo que se transmite al sistema de control y éste a su vez trata de

compensarlo rápidamente. En este sentido, el control tolerante pasivo consiste en un diseño robusto del sistema de control realimentado para hacerlo inmune a determinados fallos [Patton 1997]. Sin embargo, la teoría de control robusta muestra que sólo existen controladores sólidos para una clase reducida de cambios en la dinámica del sistema provocados por los fallos [Puig, V., Quevedo, J., Escobet, T., Morcego, B.; Ocampo, C. 2004].

El control tolerante activo consiste en el diagnóstico en línea del fallo determinando el componente que falló, el tipo, su tamaño e instante de su aparición, y a partir de esta información activa algún mecanismo de adaptación o de reconfiguración del control, dependiendo de la gravedad realiza un paro en el sistema. La adaptación al fallo reside en resolver el problema manteniendo la estructura del controlador y modificando solamente los parámetros. La reconfiguración consiste en cambiar las entradas y salidas del controlador así como reajustar la ley de control. La utilización de una u otra técnica dependerá de los objetivos de control planteados y del fallo que se presente en el sistema [Ors Carot R, Serrano J. 2000].

Desde el punto de vista de teoría de sistemas, el control tolerante a fallos trata de la interacción entre un sistema dado y su control. El término control debe considerarse en este caso en un sentido global, involucrando a la ley de control realimentado como aspectos de toma de decisiones que determinan la configuración del control.

Un problema de control estándar [Puig V. 2004] tiene como objetivo diseñar una ley de control $\boldsymbol{\upsilon}$, partiendo de un conjunto de objetivos $\boldsymbol{O}$ y un conjunto de restricciones $\boldsymbol{C}$, que describen el comportamiento dinámico del sistema (modelo matemático). El conjunto de restricciones $\boldsymbol{C}$, están determinadas por la estructura de un modelo matemático $\boldsymbol{S}$ y sus parámetros θ.

$$\boldsymbol{\upsilon = solucionar\ \{O, C(\theta)\}} \qquad \boldsymbol{(3.1)}$$

Difícilmente un modelo matemático representa convenientemente el comportamiento del sistema. Debido a problemas de perturbaciones, ruido en las medidas, dinámicas no modeladas, parámetros variantes en el tiempo e inciertos, etc., la solución al problema de control para conseguir los objetivos $\boldsymbol{O}$, cuando se considera la incertidumbre en el modelo, se puede plantear suponiendo una estructura fija $\boldsymbol{S}$ para el mismo, con parámetros θ desconocidos y que pertenecen a un conjunto de parámetros Θ, aplicando técnicas de control robusto o adaptativo [Vicenç Puig 2004].

El control robusto trata de diseñar una ley de control $\boldsymbol{\upsilon}$ que cumpla todo el conjunto de objetivos $\boldsymbol{O}$ teniendo cuenta todo el conjunto de parámetros Θ.

$$\boldsymbol{\upsilon = solucionar\ \{O, C(\Theta)\}}\ \Theta \text{ es todo el conjunto de posibles parámetros} \qquad \boldsymbol{(3.2)}$$

El control adaptativo soluciona el problema estimando a cada iteración el valor de los parámetros θ del conjunto de posibles parámetros del sistema Θ [Vicenç Puig 2004].

$$\boldsymbol{\upsilon = solucionar\ \{O, C(\theta)\}}\ \theta \text{ pertenece } \Theta \text{ se estima a cada iteración} \qquad \boldsymbol{(3.3)}$$

La aparición de fallos en el sistema puede ocasionar modificaciones en las restricciones $\boldsymbol{C}$ así como cambios en los parámetros θ haciendo que el problema de calcular la ley de control $\boldsymbol{\upsilon}$ no tenga solución a no ser que se modifique el conjunto de objetivos $\boldsymbol{O}$.

El funcionamiento del sistema realimentado se puede describir mediante las variables x_1 y x_2, como se muestra en la figura 3.2 se muestra las diferentes regiones que se deben de considerar en el diseño de un control tolerante a fallos. La región de comportamiento deseado es aquella en la que el sistema debe

operar normalmente cumpliendo su función. El controlador se encarga de mantener sistema en dicha región a pesar de las perturbaciones e incertidumbre en el modelo utilizado [Vicenç Puig 2004].

Para el diseño del lazo de control, donde se puede incluir pequeños fallos, aunque esta no es su función principal. La región de comportamiento degradado intercepta con la región de funcionamiento en la que el sistema se desplaza después de la aparición de un fallo. En esta situación el controlador tolerante diseñado para reaccionar frente a este debe de activar las acciones de recuperación con el propósito de evitar una mayor degradación de tal manera que el sistema no se desplace hacia la región de comportamiento inadmisible e incluso de peligro [Ors Carot R, Serrano J. 2000].

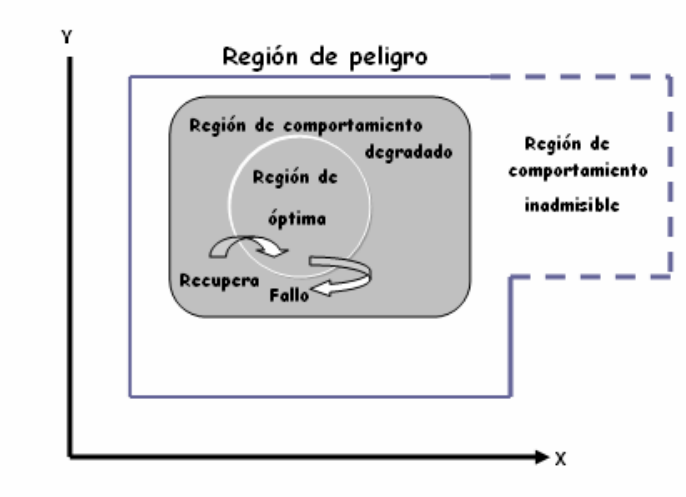


Figura 3.2 Región de comportamiento del sistema ante un fallo.

Tomando en cuenta lo descrito anteriormente se puede decir que la implementación de tolerancia a fallos en un sistema o proceso tecnológico complejo que puede ser entendido, como la capacidad con la que cuenta el sistema de control para mantener los objetivos a pesar de la aparición de un fallo, aceptando una cierta degradación en sus prestaciones.

3.2.1 Metodología para realizar el diseño de un sistema tolerante a fallos

Las etapas de la metodología de diseño del control tolerante a fallos se muestran en la figura 3.3, y se enumeran a continuación [Blanke 2000]:

1. *Análisis del Sistema* en dos niveles: a nivel de sus componentes mediante un análisis de propagación del fallos a través de todos los subsistemas más relevantes, así como una evaluación de la inflexibilidad de los mismos, y a nivel de la estructura con el propósito de analizar la redundancia presente en el sistema que ayudan en el diseño del sistema de diagnóstico.
2. *Diseño del Sistema de Diagnóstico* a partir del análisis estructural y teniendo en cuenta las medidas disponibles y los fallos que se desean diagnosticar. En el caso de que no se puedan diagnosticar todos los fallos, se deberá modificar la instrumentación disponible hasta conseguirlo. El sistema de diagnóstico de fallos debe detectar, aislar los fallos y también estimar su tamaño.
3. *Diseño de los Mecanismos de Tolerancia* para cada uno de los fallos considerados ya sea fallos en sensores, actuadores etc.
4. *Diseño del Supervisor* a partir de la información acerca de los fallos proporcionados por el sistema de diagnóstico, el supervisor debe de activar los mecanismos de tolerancia que se han diseñado para cada uno de ellos.

5. *Aplicación y test* en simulación y sobre el sistema real.

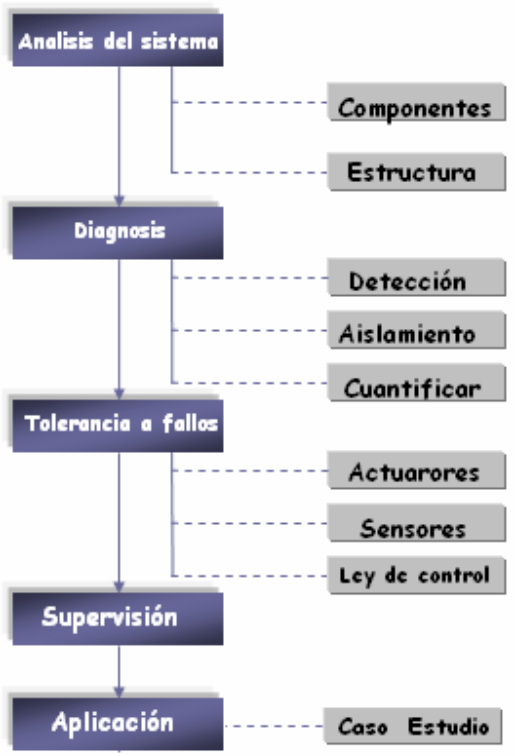


Figura 3.3 Diagrama de bloques de las etapas de la metodología par el sistema de control tolerante a fallos.

3.2.1.1 Análisis del sistema

El diseño del sistema de diagnóstico y diseño de los mecanismos de tolerancia a fallo obligan a realizar un análisis del sistema:

- ***A nivel de componentes*** mediante el análisis de propagación de fallos a través de todos los subsistemas con mayor relevancia. El resultado de este análisis permitirá identificar y listarlos componentes más relevantes, así como una evaluación minuciosa de los mismos
- ***A nivel de la estructura*** con el propósito de analizar la redundancia presente en el sistema que ayudará diseñar el sistema de diagnóstico.

En ambos casos se analiza el sistema partiendo del nivel más bajo, utilizando el concepto de componente, entendiendo como componente la unidad física considerada como indivisible, por ejemplo, un sensor, una bomba, una tubería, el controlador [Ors Carot R, Serrano J. 2000]

A nivel de componentes

El Análisis de propagación de fallos (FPA), propuesto por [Blanke 2000], tiene como objetivo estudiar cuál es el efecto final de un fallo en cada uno de los componentes del sistema de control para diseñar mecanismos de tolerancia que eviten contaminar todo el sistema. Si como resultado del análisis de

propagación de fallos en determinados componentes que pueden ser críticos, éstos pasarán a constituir la lista de fallos a detectar, aislar y tratar mediante mecanismos de tolerancia.

El método de FPA tiene como punto de partida la técnica de modos de fallos y análisis de efectos (failure mode and efect analysis, FMEA) de componentes [Herrin 1981] Se trata de un estándar comúnmente aceptado en la industria que permite analizar la propagación de los efectos de los fallos desde los componentes hacia el sistema.

Con el método FMEA se realiza una lista para cada componente en sus posibles y diferentes modos de fallo y sus efectos. El resultado del análisis nos proporciona, qué efecto produciría la aparición de de un componente en modo de fallo sobre el sistema y su entorno.

A nivel estructural

En este caso se analizan las características estructurales ***S*** del modelo, características que son independientes del valor de los parámetros. Es con la finalidad de representar las relaciones existentes entre las variables y los parámetros resultantes del modelo. El análisis permite obtener información útil para el diseño del sistema de diagnóstico así como de los mecanismos de tolerancia, permitiendo:

- Identificar aquellos componentes que se pueden monitorear.
- Obtener relaciones de redundancia analítica que permitirán la detección y aislamiento de los fallos.
- Identificar posibles mecanismos de tolerancia a fallos.

Cuando se presenta un fallo en un componente, el análisis estructural se ve modificado ya que una variable puede pasar de conocida a no conocida, o algunas de las relaciones primarias pueden dejar de ser válidas suprimiendo o no a las variables asociadas. Para que el sistema sea tolerante a fallos y detectable al mismo tiempo una vez detectado un fallo, el grado de redundancia debe ser superior o igual a 2 [Vicenç Puig 2004].

3.2.1.2 Diseño del sistema de diagnóstico de fallos

Frecuentemente se confunde los términos *detección y diagnóstico de fallos*, y es muy importante diferenciarlos pues el propósito de la detección concierne a los efectos producidos por los fallos en las variables medidas y el propósito del diagnóstico es la identificación y localización de sus causas.

La ejecución correcta de detectar y diagnosticar fallos reside en la utilización del conocimiento sobre la planta. Para la detección de fallos es suficiente conocer las condiciones de su funcionamiento normal, para el diagnóstico se requiere conocimiento profundo del proceso, incluyendo cuando se opera en fallo.

Los *mecanismos de detección* de fallos utilizados actualmente en la industria consisten en:

- Instalación de sensores redundantes en puntos críticos con la finalidad de validar la medida o detectar fallos.

- Fijar umbrales de activación de alarmas, ya sean absolutos, relativos o de relación de cambio.

Con respecto al caso del diagnóstico, son pocas las plantas que incorporan mecanismos para la identificación y localización de fallos una vez que se ha detectado el fallo. El problema principal reside en transferir el conocimiento y experiencia a las máquinas, y en cómo debe ser procesado este conocimiento.

Actualmente las técnicas de detección y diagnóstico de fallos se enfrentan al problema de complejidad de los procesos debido: a que los sistemas de control no son lineales, son cambiantes, cuentan con múltiples interacciones y acoplamientos, y tiene una gran distribución de sus tareas, etc. Esto hace difícil el desarrollo de modelos analíticos (incertidumbre en los parámetros, modelos incompletos), así como, la obtención descripciones lógicas completas de los procesos (difícil descomposición funcional, desconocimiento interno, etc.), o el establecer relaciones infalibles entre las medidas y sus causas (explosión combinatoria de posibles situaciones, ruido en las medidas, perturbaciones).La solución de los anteriores problemas ha sido la aplicación de técnicas de Inteligencia Artificial (IA), por lo que la incertidumbre y la imprecisión pueden tratarse con lógica difusa, redes neuronales artificiales ofreciendo una solución a la identificación de modelos no lineales, los algoritmos genéticos han permitido la adaptación para optimizar determinadas tareas, los sistemas expertos permiten representar y procesar relaciones heurísticas en sus reglas, el razonamiento basado en casos ofrece una forma de aprendizaje constante a través de experiencias previas. La utilización de una o varias de estas técnicas de la Inteligencia Artificial aplicadas directamente o en combinación con los métodos anteriores de detección y diagnóstico permiten abordar soluciones que hasta el momento no se podían tratar [Meléndez J., Colomer J. 2003].

De tal manera que el sistema de diagnóstico de fallos debe de realizar las siguientes tareas:

- Detectar los fallos: decidir si existe o no un fallo así como determinar el instante de su aparición.
- Aislar el fallo: localizar el componente en el cual se ha producido el fallo.
- Identificación y estimación del fallo: identificación del modo de fallo y la estimación de su magnitud.

Técnicas de detección y diagnostico de fallos basadas en modelos (MBD)

En los años 70, tal como aparece en [Frank, P.M., Ding S.X., Köppen-Seliger B. 2000], se inicia la detección y diagnóstico de fallos basado en modelos como una aplicación de la teoría de observadores utilizados en el área de control automático. Durante los siguientes 25 años la comunidad de control automático, ha realizado numerosas contribuciones en este campo. Paralelamente, en los últimos años ha habido un crecimiento importante de aportaciones procedentes de ciencias de la computación y de la comunidad de Inteligencia Artificial (esta comunidad utiliza las siglas DX para referirse al diagnóstico basado en modelos) [Puig V., Quevedo J., Escobet T., Morcego B., Ocampo C.2004].

Detección de fallos basada en modelos cuantitativos. En el caso de utilizar modelos cuantitativos, una forma de verificar la consistencia entre el modelo y las medidas de las entradas/salidas es generar, a partir de las mismas (**u, y**) y del modelo, una estimación de las salidas **y**. La consistencia entre el sistema real y el modelado se evalúa a cada instante de tiempo mediante la diferencia conocida como *residuo*, o bien, a partir de una relación más compleja que involucra a las entradas **u(k)** y salidas medidas **y(k)**, así como los parámetros del sistema denominada relación de *redundancia analítica.*

$$r(t) = y(t) - y(t) \qquad (3.4)$$

$$\varphi(t) = f\,(u(t)\ y(t), \theta) \qquad (3.5)$$

Las técnicas más utilizadas para generar residuos mediante modelos analíticos son:

- Ecuaciones de paridad [Gertler 1991].
- Observadores [Chen, J., Patton, R.J. 1999].

Las técnicas de generación de relaciones de redundancia analítica más conocidas son :

- Espacios de paridad [Chow 1984].
- Análisis estructural [Starroswiecki 2004].

Últimamente se han llegado a obtener equivalencias entre las diferentes técnicas de generación de residuos y de relaciones de redundancia analítica [Gertler 1991], [Ding 1999].

En la ausencia de fallos [Puig V., Quevedo J., Escobet T., Morcego B., Ocampo C.2004], el residuo debe ser siempre cero, mientras que en presencia de un fallo el residuo debe ser diferente de cero. Por tanto, en condiciones ideales, el test de detección de fallo consiste en comprobar si el *residuo r(t)* es nulo o no. Sin embargo, la presencia habitual de perturbaciones, ruido y errores del modelo provocan también que los residuos no sean nulos, interfiriendo en la detección posibles fallos. Por lo que se requiere diseñar residuos que estén afectados lo menos posible por las perturbaciones, ruido y dinámica no modelada, es decir, robustos frente a dichos efectos. La robustez en detección de fallos se puede alcanzar en la generación de los residuos (robustez activa) o en la fase de toma de decisión (robustez pasiva).

Para validar la teoría del control robusto en las décadas de los 80's y 90's [Puig V., Quevedo J., Escobet T., Morcego B., Ocampo C.2004]se realizaron búsquedas de técnicas activas de diagnóstico robusto. Destacándose aportaciones como la utilización de conocimiento estadístico de perturbaciones [Basseville 1993]; la utilización de funciones de transferencia para el cálculo de los residuos que permite un desacoplo respecto de las perturbaciones [Gertler 1998], [Chen 1999] o la utilización de algunos índices de prestaciones para generar los residuos como propone [Frank, P.M., Ding S.X., Köppen-Seliger B. 2000].

La robustez pasiva [Puig V., Quevedo J., Escobet T., Morcego B., Ocampo C.2004] consiste en tener en cuenta la incertidumbre en la generación de los umbrales que han de superar los residuos para considerar la existencia de fallos. Estos umbrales por lo general dependen de las condiciones de operación del proceso controlado y si está en estado transitorio o permanente, por lo que es posible determinar empíricamente o analíticamente umbrales adaptativos que permitan realizar una detección robusta de fallos. Las técnicas de umbrales adaptativos fueron propuestas inicialmente por [Clark 1989], que sugirió una relación empírica entre el punto de operación y el correspondiente umbral para la detección. Otra técnica es la propuesta por [Emami-Naeini, A., Akhter M.M., Rock S.M. 1988] que desarrolló una relación teórica, basada en H1, entre el punto de operación, la incertidumbre del modelo y el umbral de detección. [Frank, P.M., Ding S.X., Köppen-Seliger B. 2000] también utiliza técnicas basadas en H1 para obtener umbrales adaptativos. Otra técnica basada en optimización dinámica asumiendo incertidumbre en los parámetros del modelo fue propuesta por [Horak, D.T., Guidance, J. 1988], y varias propuestas utilizando modelos intercalares en simulación, predicción y observación han sido descritas en [Puig V. 2002a].

Detección de fallos basada en modelos cualitativos. En ciertos casos es difícil disponer del conocimiento completo del proceso para construir un modelo analítico suficientemente representativo del mismo y, por lo tanto, se puede hacer inevitable grandes desviaciones entre la realidad y el modelo que hagan inservible este procedimiento para diagnosticar fallos. De forma alternativa, un conocimiento

incompleto puede ser tratado de forma abstracta mediante un modelo cualitativo, que enfatice distinciones y relaciones primarias del proceso e ignore relaciones no importantes o desconocidas.

Aunque los modelos cualitativos son por naturaleza imprecisos, pueden estar aptos para representar bien el comportamiento del proceso complejo. En este caso, se utilizan conjuntos de valores catalogados mediante un atributo (positivo, negativo, disminuye, etc.) en lugar de valores numéricos como elementos de base para la representación de modelos cualitativos. Otro tipo de modelos son los semicualitativos utilizan conjuntos de valores caracterizados por intervalos o por conjuntos difusos [Puig V., Quevedo J., Escobet T., Morcego B., Ocampo C.2004].

En los últimos años, el estudio de modelos cualitativos o semicualitativos para monitoreo y diagnóstico de fallos está teniendo una gran respuesta [Kuipers 2006], [Leitch R. 2003], [Travé, Massuyes 2001]. Los descriptores cualitativos de las variables pueden ser signos [De Kleer 1984], intervalos [Puig V. 2002a] o conjuntos difusos [Shen 1993]. Inclusive, los conjuntos difusos pueden ser una serie de intervalos, utilizando el principio de identidad **α-corte** reduciendo el tratamiento difuso en cálculo intercalar [Puig V. 2002b].

Entre las técnicas más relevantes que utilizan modelos cualitativos o semicualitativos para el diagnóstico de fallos se pueden citar:

-Observadores cualitativos utilizan ecuaciones diferenciales cualitativas (QDE). Siendo estas una extensión de las ecuaciones diferenciales ordinarias que utilizan variables, parámetros y funciones imprecisas que se pueden representar mediante reglas ***If Then***.

-Detección de fallos utilizando envolventes que engloban la respuesta de la familia de sistemas representados por modelos cualitativos que utilizaban intervalos de valores en lugar de valores numéricos simples (modelos semicuantitativos). En [Bonarini, A., G. Bontempi 1994], [Puig V. 2003], los parámetros y las variables de estado intercalares son tratadas como un hipercubo que evoluciona a lo largo del tiempo y que proporciona unas envolventes adaptativas donde debe estar siempre la respuesta del sistema real [Puig V., Quevedo J., Escobet T., Morcego B., Ocampo C.2004].

Técnicas de detección y diagnostico de fallos mediante inteligencia artificial

Uno de los aspectos fundamentales para poder utilizar técnicas de Inteligencia Artificial para el análisis de situaciones en procesos industriales es el tratamiento de las señales provenientes del proceso. Se trata, en definitiva, de extraer y codificar, *a partir de las señales*, aquella información útil sobre los fallos que deben detectarse o diagnosticarse. Si el objetivo es evaluar estas señales para decidir sobre el estado del proceso, deben establecerse los mecanismos que permitan tratar los diversos problemas que pueden afectarlas, como pueden ser la imprecisión, la incertidumbre, la ausencia o la cantidad excesiva de información. El tipo de procesado a utilizar dependerá del tipo de señales y del propio conocimiento del proceso que posteriormente va a ser utilizado para la detección o el diagnóstico. Las técnicas utilizadas, por lo tanto, son muy variadas [Ors Carot R, Serrano Juan J.20003].
.

En detección y diagnóstico, las conclusiones o decisiones difícilmente pueden tener estrictamente de la forma sí o no, verdadero o falso. La complejidad del conocimiento del proceso, o las condiciones vagas, inciertas, siempre llevan conclusiones intermedias, transitorias o vagas.

Para representar los datos, los conjuntos difusos permiten cuantificar esta vaguedad definiendo funciones de pertenencia asociadas a éstos para designar el grado en que un elemento pertenece a un

conjunto. La teoría de la lógica difusa da una solución al problema de la representación del conocimiento y del razonamiento con datos imprecisos.

Se ha utilizado a la lógica difusa en la evaluación de residuos para el diagnóstico de fallos, o en redes neuronales entrenadas para simular el comportamiento del proceso o para clasificar determinadas situaciones.

Las redes neuronales artificiales (RNA) tienen aplicaciones en la detección de fallos, debido a su capacidad de aproximar funciones no lineales, y se han venido utilizando en sustitución de los modelos analíticos para simular sistemas altamente no lineales como son las plantas químicas, nucleares, sistemas de control de vuelo, etc.

El uso de RNA para el diagnóstico de fallos puede encontrase en la implementación de algoritmos de reconocimiento de patrones correspondientes a situaciones de fallos, operando directamente sobre las señales adquiridas, o como sistema de evaluación de los residuos sensibles a los diferentes modos de fallo en estrategias basadas en modelo.

La utilización de RNA como clasificador supervisado permite la estimación de índices no mesurables o la asociación automática de síntomas (extraídos de las señales) con diagnósticos. Asimismo, se utilizan para descubrir y clasificar comportamientos característicos de los procesos. El siguiente ejemplo es el de una RNA para la localización de fallos en sistemas de distribución eléctrica, donde se utiliza una RNA de dos capas para determinar el origen de perturbaciones en la red eléctrica.

Se entrenó una RNA basándose en las medidas tomadas en una subestación y caracterizadas por dos parámetros, la tensión característica y el factor PN, de manera que es posible asociar a una nueva perturbación un origen en el sistema eléctrico (Distribución o Transmisión). La figura 3.4 muestra el error para diferentes datos de entrenamiento y el número de neuronas en la capa oculta [Meléndez J., Colomer J. 2003].

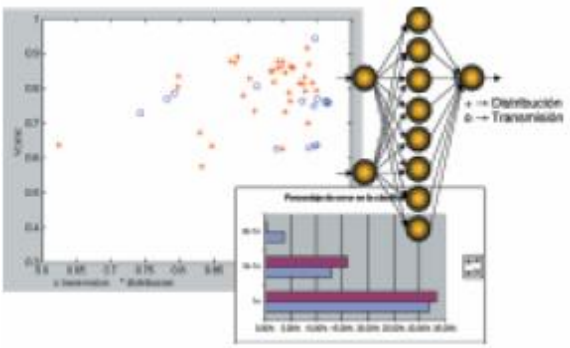


Figura 3.4 RNA utilizada en la detección y localización de fallos.

Los métodos basados en análisis causal utilizan las relaciones síntoma-diagnóstico o relaciones causa-efecto (modelado causal). Las primeras aplicaciones basadas en estos métodos se llaman árboles de fallos (figura 3.5), formados por nodos que representan hechos y puertas lógicas que codifican las relaciones entre éstos. Los hechos se conectan mediante flechas de unión; un nodo tipo OR representa la disyunción de condiciones que causan una salida, mientras un nodo tipo AND representa la conjunción de condiciones que causan la salida.

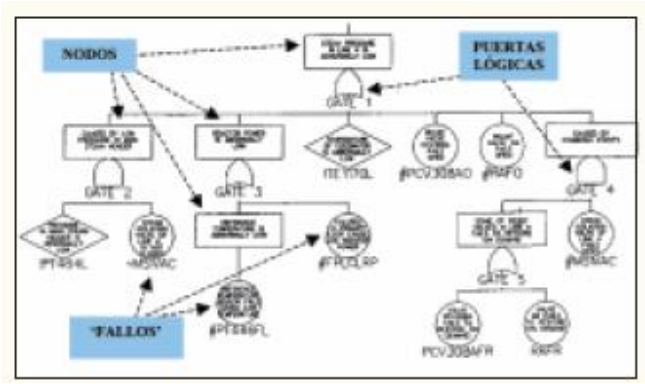


Figura 3.5 Árbol de fallos extraído de Reliability Engineering and System Safet, núm. 44, 1994.

Otra herramienta es la denomina ***Signed Directed Graph (SDG***), establecida para modelar procesos continuos. Un SDG consiste en nodos que representan variables numéricas o de tipo cualitativo (por ejemplo: normal, alto, bajo) y las ramas representan relaciones causales entre los nodos que pueden ser positivas (+) o negativas (-). Cada nodo puede influir sobre uno o varios nodos. Las relaciones causales pueden incluir información auxiliar, como retardos, intensidad, probabilidades, etc. La idea básica de esta técnica consiste en rastrear los funcionamientos defectuosos. Puede usarse tanto para el modelado de fallos (propagación de fallos) como para el diagnóstico (buscando causas iniciales de los funcionamientos defectuosos observados, desde los síntomas hasta el origen del fallo). Para casos prácticos de aplicación, los grafos pueden ser representados de forma equivalente por un conjunto de reglas [Meléndez J., Colomer J. 2003].

Los sistemas expertos intentan codificar el conocimiento de un experto humano mediante un sistema basado en reglas de producción de forma que puedan expresarse la experiencia, el razonamiento aproximado, la imprecisión, el razonamiento por defecto, aprendizaje, etc.

Específicamente, se trata de representar el conocimiento experto para tener un sistema que responda como lo haría el experto humano. En su forma tradicional, un sistema experto está formado por una base de reglas y un motor de inferencia que razona sobre una base de hechos. Una base de reglas es un conjunto de reglas del tipo *if then*. *Lo verdadero* significa la seguridad con que el experto del proceso hace esta afirmación que es convertida en una regla. La base de hechos es el conjunto de evidencias, junto con sus certezas asociadas como una variable de medida (ejemplo la temperatura de una caldera) o una posible alarma (Alarma 23 con certeza 30%) o conclusión de una o varias reglas (Diagnostico: Problemas con la válvula 12, con certeza 90%).

El motor de inferencia se encarga de recorrer las reglas inspeccionando si las puede aplicar, esto es, que se encarga de ejecutar el razonamiento. Existen diversas formas de realizar dicha ejecución, por lo que el razonamiento consiste en aplicar una base de reglas a una base de hechos y de esta manera obtener nuevas conclusiones. A medida que se van aplicando las reglas se deducen nuevos hechos que se añaden a la base de hechos.

Los sistemas expertos son capaces de manipular información incierta o de tipo cualitativo, permitiendo de esta forma codificar conocimientos inciertos o imprecisos. Las ventajas de la utilización de sistemas expertos para detectar y diagnosticar fallos residen en la posibilidad de representar y codificar el conocimiento experimental, mientras que el principal inconveniente es la obtención de este tipo de conocimiento. Su campo de aplicación se reduce a instalaciones complejas y normalmente de grandes dimensiones (procesos petroquímicos y cementeros, etc.).

La gran mayoría de las aplicaciones actuales de detección y diagnóstico de fallos van asociadas a resultados de investigación y son soluciones a la medida. La tendencia es desarrollar sistemas integrados que deben incluir y combinan las diversas técnicas.

Cada técnica para la detección y el diagnóstico de fallos presentada anteriormente tiene sus ventajas y desventajas Por lo tanto, la combinación de dos o más técnicas puede, muchas veces, resolver problemas difíciles. Algunas aplicaciones, combinan simulaciones del proceso con sistemas basados en reglas permitiendo así la incorporación del conocimiento de los expertos y sistemas de razonamiento basado en casos que permiten aprovechar los datos históricos En otros casos, las redes neuronales se usan para modelar sólo las partes desconocidas de modelos físicos, para mejorar los métodos estadísticos (permitiendo no linealidades) o como parte de sistemas expertos de diagnóstico. Actualmente, las tareas directamente relacionadas con la adquisición y almacenamiento de datos, la representación gráfica y animada de variables de proceso, y monitorear de éstas se realizan mediante los sistemas Supervisor y Control y Adquisición de Datos (SCADA), que permiten además actuar sobre autómatas y reguladores autónomos o directamente sobre el proceso[Ors Carot R, Serrano J. 20003]
.

3.2.1.3 Diseño de los mecanismos de tolerancia a fallos

La estrategia de tolerancia a fallos que se requiere aplicar va a depender del componente o del lazo de control que se vea afectado por el fallo. En primer lugar, si el proceso controlado dispone de redundancia física, en más de un componente (sensor, actuador, elemento del proceso) para realizar la misma función, entonces la estrategia de tolerancia a fallos consistirá solamente de sustituir el componente en fallo por otro igual que funcione bien. Esta estrategia es costosa ya que se duplica o triplica los componentes críticos de un proceso controlado, y no siempre es posible incorporarlos físicamente en espacios reducidos [Puig V., Quevedo J., Escobet T., Morcego B., Ocampo C.2004].

Cuando no existe redundancia física, se debe de distinguir entre fallos en sensores, actuadores y en la propia planta. Para adaptar los fallos en los sensores se utilizan los *sensores virtuales* que estiman la medida del sensor en fallo a partir del resto de los sensores que existen en el sistema. Para adaptar los fallos en actuadores se opta por el rediseño de controladores, utilizándose principalmente dos mecanismos: la adaptación al fallo y la reconfiguración, según se cambie la ley o la estructura de control, respectivamente [Blanke, M., Kinnaert M. Lunze J. y Staroswiecki M. 2003].

La incorporación de mecanismos de control tolerante en el lazo de control depende del tipo de control utilizado [Puig V. 2001]. Así, por ejemplo, existen estrategias de control, como el control predictivo donde se añaden nuevas restricciones al problema de optimización, permitiendo fácilmente incorporar mecanismos de tolerancia a fallos [Maciejowski 2001]. Debido a la aparición de fallos en el sistema y la activación de mecanismos de tolerancia a estos se pueden modelar como eventos discretos en el tiempo que provocan cambios en el comportamiento dinámico continuo del sistema, los sistemas de control tolerantes a fallos son de naturaleza híbrida. Por lo tanto, para el análisis y diseño de controladores tolerantes se debe utilizar técnicas desarrolladas para sistemas híbridos [Cassandras, C.G., Lafortune S., Olsder G.J. 1995], [Morari 2003].

Por lo anteriormente expuesto se puede decir que dentro de un lazo de control tolerante se debe de considerar que existe tolerancia a fallo al contar con:

- Mecanismos que introducen redundancia en los sensores y/o actuadores.
- Estrategias de adaptación de la ley de control que gobierna el lazo.

Tolerancia por reposición de sensores y/o actuadores

Fallos graves en sensores o actuadores rompen los lazos de control. Para mantener el sistema control en funcionamiento es necesario utilizar diferentes actuadores (entradas) y/o sensores (salidas). Utilizando un bloque de reconfiguración que junto con la planta en fallo, la planta reconfigurada se comporte igual que la planta sin fallo. Esta solución trata de aplicar los mínimos cambios al lazo de control, de tal manera que el controlador estándar pueda continuar controlando la planta como si no existiera fallo alguno [Puig V., Quevedo J., Escobet T., Morcego B., Ocampo C.2004].

Para cualquier caso, ya sea en sensores o actuadores, se maneja la idea de reconfiguración desde los siguientes puntos de vista:

- Mediante redundancia física o redundancia en hardware.
- Mediante redundancia analítica o redundancia de software.

Mediante redundancia física. En el caso de los sensores, consiste en contar con un número generalmente impar de ellos, cuyas salidas se multiplexan dentro de un bloque de decisión. En este bloque se determina la medida correcta a partir de la salida más común producida por cada sensor. El caso de actuadores es más directo, al producirse un fallo, la redundancia física implica contar con otro dispositivo de respaldo listo para realizar la acción de control previa determinación de daño en el actuador principal.

Mediante redundancia analítica: Consiste en la incorporación al lazo de un bloque que reconstruya las medidas mediante la estimación de las mismas (para sensores) o la idea de reajuste de señales alternativas para llevar a cabo la acción de control requerida (para actuadores). Así se evita la incorporación de nuevo hardware en el sistema lo que se ve reflejado en costos de instrumentación [Puig V., Quevedo J., Escobet T., Morcego B., Ocampo C.2004].

Tolerancia por adaptación de la estrategia de control

En esta estrategia se consideran dos grupos principales: las técnicas activas y las pasivas.

Técnicas pasivas son leyes de control que toman en cuenta la posible aparición de un fallo tratado como una perturbación al sistema. De tal manera que dentro de ciertos márgenes de capacidad, la ley de control se adapta de tal manera que el sistema admite la presencia del fallo sin tomar en cuenta a un sistema de diagnóstico. Esto lo hace bastante restrictivo para soportar fallos de magnitud importante o fenómenos dinámicos no considerados dentro del diseño.

Técnicas activas consisten en reconfigurar la ley de control basándose en el uso de un *diagnosticado*r que provee la información necesaria para realizar automáticamente los ajustes necesarios con la finalidad de cumplir con los objetivos de control. El sistema de diagnóstico de fallos informa qué restricciones han cambiado y qué leyes de control ya no se pueden utilizar, se deben de considerar dos casos, dependiendo de si el algoritmo de diagnóstico ha sido capaz de proporcionar:

- Estimar el impacto del fallo.
- Solamente la detección y el aislamiento del fallo sin estimar la magnitud de su impacto.

Existen dos formas de rediseñar el sistema de control con el propósito de introducir tolerancia a fallos dependiendo de sus efectos o si se ha producido cambios en la estructura del sistema, siendo este:

- Cambiando la ley de control sin cambiar elementos del lazo de control mediante adaptación al efecto del fallo, o el caso que se haya podido estimar los cambios de estructura y parámetros que se introdujeron debido al fallo.
- Cambiando la ley de control y los elementos del lazo mediante su reconfiguración frente al fallo, en caso de que no se haya podido estimar los cambios de estructura y parámetros introducidos por el fallo. En éste caso, se obliga a desconectar los componentes en fallo localizados por el sistema de diagnóstico, y se deben de alcanzar los objetivos de control utilizando solo los componentes que no presentan fallo.

La diferencia principal entre estas estrategias radica en que en el caso de la *reconfiguración*, se utilizan diferentes señales de control, y medidas entre el controlador, y la planta, siendo un cambio en su estructura, mientras que mediante la *adaptación* se pretende que sea la propia ley de control la que compense el fallo.

Tolerancia a fallos en actuadores. Consiste en la capacidad que tiene el lazo de control de soportar la influencia de un fallo originado en la etapa de ejecución de las acciones determinadas por el controlador. Estas acciones, por estar relacionadas con actuadores físicos, pueden ser irremplazables y un fallo origina la incapacidad del sistema para ejecutar cualquier acción, lo que en la práctica es una necesidad apremiante de incorporar al menos otro actuador redundante. Sin embargo y en el área del análisis teórico, se ha propuesto recientemente una estrategia que se muestra cómo dual del sensor virtual conocida como actuador virtual Esta estrategia supone un modelo nominal del proceso, y un modelo de la planta cuando se presenta fallo en el actuador. Mediante la inclusión de un observador y un controlador por realimentación de estados se pretende cumplir los siguientes objetivos:

1- Que el controlador haga que el lazo de control reconfigurado se comporte como el modelo nominal del sistema, lo que es lo mismo $x_f(t) = x_n(t)$ (Strong Reconfiguration Goal).
2- Que la salida del sistema reconfigurado tienda al valor de salida del modelo nominal en presencia de una señal de consigna constante, es decir $y_f(t) \to y_n(t)$ (Weak Reconfiguration Goal).

La condición fundamental para la aplicación de estás técnicas, y para asegurar la estabilidad del sistema reconfigurado es que éste modelo del sistema en fallo sea controlable. En estas condiciones, dado un modelo en fallo (contemplada la restricción anterior), un controlador nominal y un punto de equilibrio se diseñan en un actuador virtual (observador y realimentación de estados) para incorporarlo al lazo de control y obtener el sistema reconfigurado capaz de tolerar fallos en actuadores.

Sin embargo, la estrategia deja cuestiones abiertas en cuanto a sus limitaciones de implementación. Entre otras se puede citar el conocimiento necesario de los efectos del fallo sobre el actuador y su modelado para obtener el modelo del sistema necesario a usar en el cálculo del actuador virtual.

Una manera más implícita de plantear la tolerancia a fallos en actuadores consiste en utilizar la información del diagnosticador para realizar las consideraciones necesarias que puedan ser incorporadas en los criterios de modificación del controlador o en su ley de ajuste [Zhang 2000], [Wang 2000] y [Tao 2002]. De esta manera el fallo ocurre en el actuador y el que asume la función de tolerancia y reposición es la ley de control, suponiendo la existencia de redundancia física o una disposición apropiada de actuadores y una combinación adecuada de comandos que, mediante relaciones indirectas, hagan que se cumpla el objetivo de control predefinido [Dardinier Maron 1999]. Estas técnicas se acercan más a la adaptación, ligada a la ley de control, que a la reconfiguración.

Otras estrategias se basan en la detección de la magnitud del fallo y el aprovechamiento de esta información utilizando el modelo nominal, el modelo en fallo y una relación definida entre ambos [Zhou 2002)] para generar el modelo del sistema. Es así como la expresión:

$$B_f = B_n(I - r(t)) \qquad (3.6)$$

donde r(t) = diag[$\gamma_1,\gamma_2,\gamma_3\ldots,\gamma_m$], realiza la menciona relación entre modelos. Los términos $\gamma_i(t)$, donde i = 1,2,3,…,n denotan el estado de cada actuador, siendo 1 si el actuador está completamente averiado, 0 si el actuador está en perfectas condiciones y $0 < \gamma i < 1$ si el actuador presenta pérdida parcial de efectividad. Por lo tanto, la estimación de la matriz r(t) se realiza en línea con la finalidad de determinar la influencia del fallo en el proceso y el modelo resultante se encuentra listo para aplicar alguna técnicas de adaptación [Puig V., Quevedo J., Escobet T., Morcego B., Ocampo C.2004].

Tolerancia a fallos sensores. En este caso el bloque de reconfiguración consiste en utilizar un observador que permita reconstruir las medidas del sistema a partir de otros sensores existentes, por lo que se denomina sensor virtual o software. Las técnicas de reposición de medidas basadas en filtros de Kalman, no sólo tienen validez en el ámbito de fallos sino también en la idea de optimizar el proceso de medir las variables dentro de un control industrial moderno. La posibilidad de estimar variables se encuentra estrechamente ligada con las especificaciones particulares del sistema y con la disponibilidad de elementos de medida. El diseño de una red de sensores teniendo en cuenta los criterios de tolerancia a fallos, observación del sistema, costos y robustez es tema de amplio estudio actualmente en la literatura [Hoblos 2000], [Attouche 2001].

Staroswiecki y sus colaboradores en el año del 2004 proponen la estimación de la tolerancia a fallos asociada al diseño de redes de sensores, analizando el tiempo de utilidad del conjunto de sensores, evaluando su capacidad de tolerancia y conjunto mínimo necesario considerando la redundancia. En el campo de las aplicaciones en este tema se pueden citar los trabajos basados en fallos de sensores en el campo de la aeronáutica [Lyshevski, S., Dunipace K.,Colgren R. 1999], [Huo 2001], y en turbinas de gas.

3.2.1.4 Diseño del supervisor

El sistema supervisor es un sistema basado en eventos discretos, mientras que el sistema supervisado es un sistema a tiempo continuo. El nivel supervisor intercambia información con el sistema supervisado mediante un diagnóstico, el cual proporciona información sobre la existencia de fallos y sus características, y actúa sobre el mismo mediante la activación de los mecanismos de adaptación y reconfiguración frente a los fallos. Debido a la naturaleza híbrida de los sistemas de control tolerante, su análisis y diseño se puede abordar mediante la teoría de sistemas híbridos [Cassandras, C.G., Lafortune S., Olsder G.J. 1995], [Morari 2003].

El sistema supervisor debe realizar una serie de funciones que se detalla a continuación para que el sistema de control alcance la tolerancia a fallos:

- Monitorear la planta a controlar.
- Detección y diagnóstico de fallos.
- Evaluación de la situación después de que un fallos ha sido detectado y diagnosticado, tomando en cuenta si se activan o no los mecanismos tolerantes a fallos.
- Ejecutar acciones de tipo correctivas que adapten al sistema al fallo permitiendo así que continué funcionando de manera degradada aceptable mientas le sea posible.
- Comunicación con el resto del sistema para intercambiar información.

La implementación del supervisor en un sistema de control requiere de una arquitectura apropiada que deberá de ser capaz de poder implementar las anteriores funciones. La figura 3.6 muestra una de varias arquitecturas existentes actualmente para un sistema de control tolerante a fallos que contiene los tres niveles descritos por: la parte del lazo tradicional de control (nivel 1), el sistema diagnóstico ya sea de tipo adaptativo o por reconfiguración al fallo (nivel 2), y el nivel de supervisión (nivel 3) que cierra el lazo exterior y añade la tolerancia frente al fallo [Blake 2003].

En la figura 3.6 [Puig V., Quevedo J., Escobet T., Morcego B., Ocampo C.2004] se define el lazo de control realimentado que consta de una ley de control, un actuador, la planta y un sensor. En paralelo con los bloques del actuador y el sensor se encuentran elementos de hardware o software encargados de proporcionar redundancia tanto en la medición de señales como en la ejecución de acciones de control. La redundancia se puede introducir de forma física (sensores o actuadores redundantes), o bien de forma analítica (mediante modelos). Partiendo de medidas de entradas y salidas del actuador, sensor y la planta, el *diagnosticador* realiza la detección y aislamiento del fallo, y si es posible, la cuantificación de su magnitud además de discriminar el elemento que se encuentra implicado. El diagnosticador al detectar, aislar y estimar el fallo, debe de comunicar al sistema supervisor automático (SA) lo ocurrido y éste se encarga de tomar las decisiones de recuperación del lazo de control frente al fallo reportado, evaluando una serie de criterios y realizando, de ser posible, un conjunto de acciones que pueden ser visualizadas en la figura 3.7.

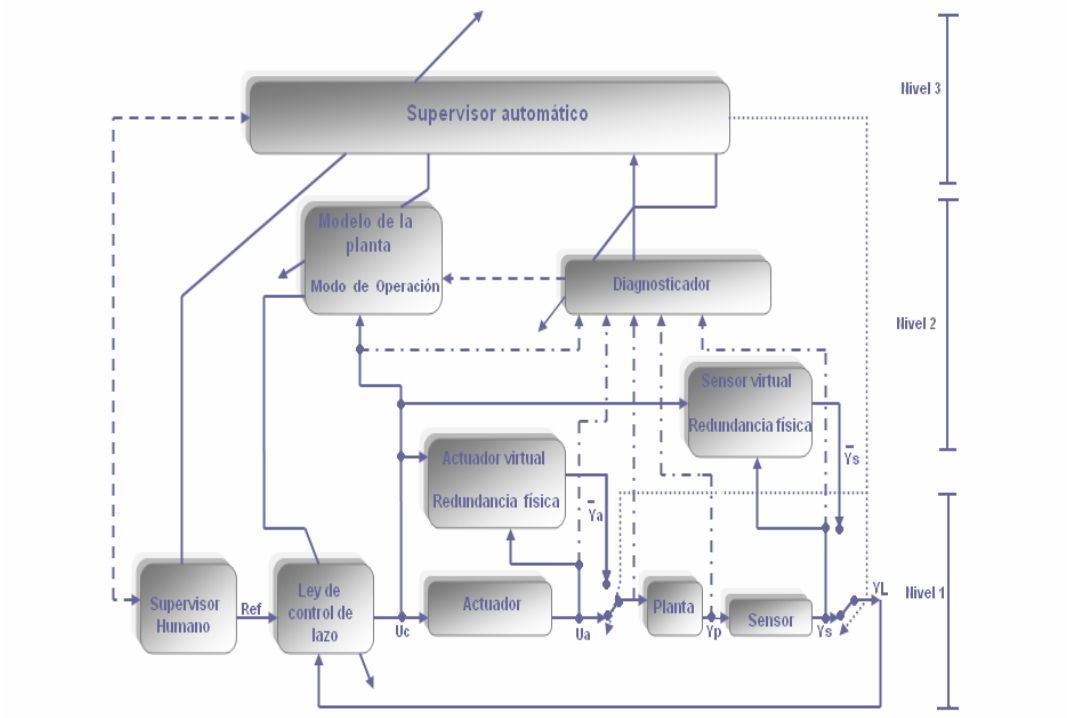


Figura 3.6 Arquitectura de un sistema de control tolerante a fallos.

El supervisor automático debe ser configurado por el supervisor humano (SH), que es un operador que impone las consignas para el lazo de control y analiza la posible información que

el supervisor automático pueda ofrecer en lo que respecta a estadística de fallos o al desempeño del sistema.

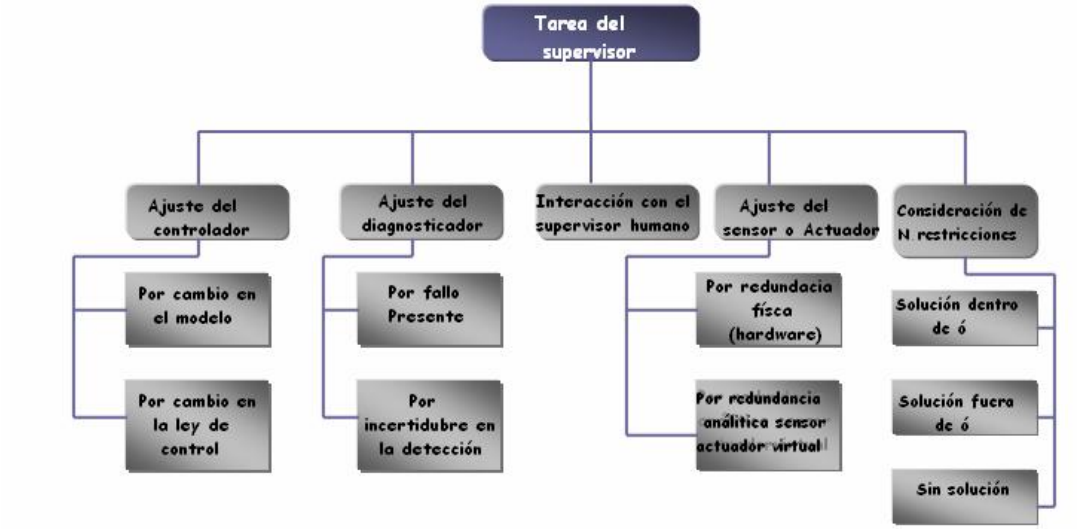


Figura 3.7 Diagrama que presenta las tareas que debe de realizar un supervisor automático.

El sistema de supervisión automático que existe comercialmente más parecido al descrito anteriormente es el denominado SCADA (Supervisory Control and Data Acquisition, y en español Control supervisor y adquisición de datos). Este tipo de supervisor es una aplicación diseñada por medio de la computadora, que incluye adquisición de datos, almacenamiento de datos, interface de comunicaciones, interface de operación y utilización de datos. El SCADA más actual ha evolucionado hacia una compleja base de datos que alimenta y es alimentada por aplicaciones interrelacionadas de gestión de Procesos por lote (batch), análisis de históricos, generación de informes, etc. Uno de los módulos que actualmente cualquier SCADA tiene incorporado es el de programación, que implica algoritmos de control y/o supervisión de la planta. Los lenguajes de programación que más se utilizan para este módulo son C, C^{++}, C^{++} VISUAL, Delphi, etc. De los *cincuenta y uno* SCADA analizados en [Ayza 2002], solo en uno declara tener un módulo de gestión de fallos. La mayoría solamente incorporan alarmas, tratando solamente detección de una determinada situación, basada en superar umbrales o activar señales binarias. Actualmente se han incorporado nuevas herramientas en los SCADA modernos que permiten al operador humano determinar el funcionamiento correcto o incorrecto de la planta, y puede tomar medidas de corrección si es necesario. Si los SCADA comerciales continúan con esta tendencia, el próximo paso será incorporar módulos de control tolerante a fallos para asistir al operador humano en sus tareas 7 [Puig V., Quevedo J., Escobet T., Morcego B., Ocampo C.2004].

Descritas de manera general, las formas más comunes de detección y diagnosis de fallos en los sistemas de control comunes, el siguiente paso es describir como se han abordado estos aspectos en el campo de la robótica móvil.

3.3 Diagnóstico de fallos en un robot móvil

En los sistemas de control de aviones, naves espaciales, plantas nucleares, plantas de procesos químicos, sistemas de control cuyo soporte es una computadora, se ha requerido implementar técnicas

tolerantes a fallos. En estas técnicas se han basado los diseñadores de algunos robots que incluyen en su modelo algunos mecanismos tolerantes a fallos.
Los robots tienen una base computacional muy fuerte, por lo que en esta sección se iniciará comentando la manera en que se toleran los fallos en sistemas computacionales, que sirven también como base para diseñar el sistema de control tolerante a fallos en los sistemas robóticos

Un método comúnmente utilizado para proporcionar tolerancia a fallos en sistemas computacionales, utiliza la redundancia modular triple (TMR), en la cual tres procesadores trabajan sobre el mismo problema y comparan sus resultados. Si uno de los procesadores esta defectuoso y su resultado no conviene con los resultados de los otros dos procesadores (ver figura 3.8), el resultado del procesador defectuoso no se toma en cuenta como correcto, y así detecta que el procesador ha fallado [Visinsky Monica L. 2003]. Solamente un procesador defectuoso puede ser tolerarlo con este método de redundancia. Sin embargo una mayor cantidad de fallos pueden ser detectados, si se incrementa el número de computadoras, siendo el caso utilizado por la NASA para tolerar los fallos en la plataforma de lanzamiento para cohetes del espacio, donde utiliza cinco computadoras redundantes (GPCs), cuatro de las computadoras son duplicadas trabajando así de manera redundante para realizar las mismas tareas con los mismos datos de entrada. Se comparan los comandos de salida, y las cuatro computadoras votan sobre los resultados y de esta manera se pueden detectar hasta dos computadoras con fallos críticos. Después de que dos computadoras han fallado, las dos computadoras restantes utilizan la comparación y métodos de autoprueba para tolerar un tercer fallo. En la quina computadora funciona el software de reserva de vuelo que realiza generalmente funciones críticas, esta quinta computadora se puede utilizar como reserva si el fallo se debe a un desperfecto en el diseño arquitectónico en el GPCs principal [Visinsky Monica L 2003].

Para evitar agregar una multiplicidad de componentes redundantes a los sistemas computacionales, se ha han desarrollado otros métodos que pueden configurar nuevamente los datos o el código dentro del computador entre las piezas que están funcionando una vez que un componente ha fallado. Algunos sistemas computacionales que poseen mecanismos tolerantes a fallos manejan un fallo permitiendo una degradación en su funcionalidad o velocidad [Ors Carot R .2004].

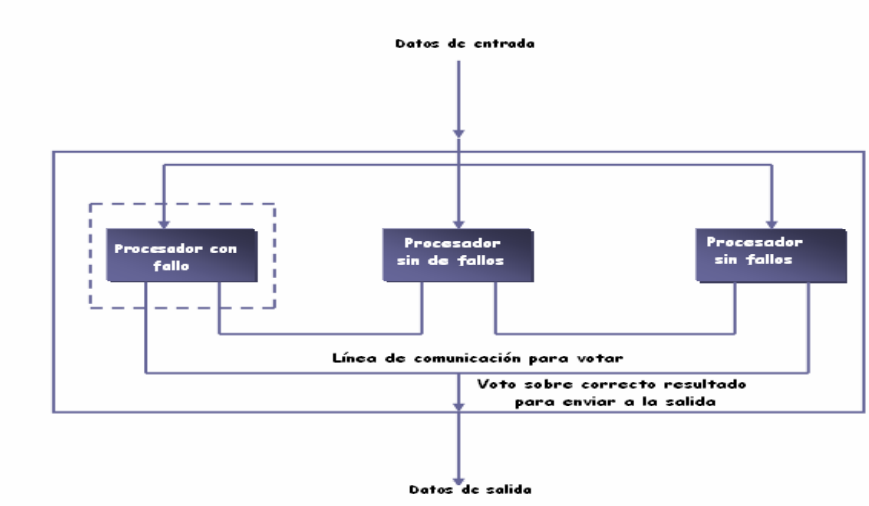


Figura 3.8 Redundancia modular triple (TRM).

Otro sistema utilizado es el esquema procesador-conmutación (set-switching) para la reconfiguración. En el esquema procesador-conmutación los componentes libres de fallos son seleccionados para

formar una sub-parte básica, tal como en una fila en un arreglo, con la configuración deseada hasta que se alcanza la configuración completamente. Este método puede requerir muchas interconexiones adicionales entre los componentes. En software, los códigos aritméticos se utilizan para encontrar y corregir errores en una matriz de cálculo computacional, este tipo de método se lleva a cabo en cinemática de los robots. Se realiza un chequeo de bit y de los códigos de corrección de errores ayudando a la transmitir los de datos del monitor y permitiendo una reconstrucción de los datos originales si la transmisión de la línea defectuosa [Visinsky Monica L 2003].

La redundancia analítica es otro concepto para la detección y aislamiento de fallos que utiliza solamente los componentes disponibles del sensor en el sistema robótico para generar los residuos por los cuales los fallos pueden ser identificados. Comparando el historial de las salidas del sensor contra las entradas del actuador, los resultados de los sensores se pueden comparar en tiempos diferentes para comprobar si hay fallos. Chow y Willsky desarrollaron un método matemático muy útil para determinar redundancias que son relevantes y se deben de considerar cuando se tiene un determinado fallo.

En el campo de la robótica móvil debido a la complejidad de los sistemas de navegación, la multitud de sensores presentes y la imposibilidad de modelar completamente el entorno en el que el robot se mueve para realizar una tarea determinada se requiere realizar un diagnóstico de fallos activo dinámicamente.

Como resultado de los avances en área de la robótica, la confiabilidad y la tolerancia a fallos están siendo factores importantes en el diseño, y se están tomando muy en cuenta con el objetivo de obtener robots confiables y robustos.

Por lo que se puede decir que un objetivo es que el robot realice su tarea de forma autónoma durante largos períodos de tiempo sin necesidad de intervención humana, o que esta ocurra lo menos posible. Debido a que es casi imposible para él diseñador predecir todas las situaciones en las que se puede encontrar el robot, es necesario establecer un mecanismo para tratar situaciones distintas de las *nominales*. Se denominan situaciones *nominales* aquellas que son contempladas a realizar por el sistema de control, y las situaciones *no nominales* (o excepciones) son el resto incluyendo, entre otras, fallos de sensores, actuadores, procesos o situaciones del entorno no contempladas. Por otra parte, si se desea lograr mayor autonomía en el robot, es necesario que su sistema sea capaz de funcionar incluso ante posibles fallos que puedan presentar alguno de sus componentes. El funcionamiento de este módulo no debe depender de otros módulos, pero sí obtiene la información acerca del estado del sistema, observando al resto y obteniendo *síntomas* acerca de situaciones de excepción [Fernández J. 1998]. La obtención de algunos síntomas es, por tanto, dependiente del sistema de control.

Los fallos más estudiados y tratados son los que suceden en el sistema sensorial del robot ya sea sólo a nivel hardware by también fallos a nivel software. Mientras algunas arquitecturas [Ferrell 1999] sólo consideran fallos en sensores y actuadores, en otras también se tienen en cuenta los fallos a nivel software especialmente en navegación [Ferrell Cinthya 1999].

El tratamiento de fallos casi siempre está siempre condicionado por la arquitectura de navegación que el robot utilice, dando lugar a diferentes sistemas de supervisión, detección y recuperación ante fallos. No sólo existe una gran variedad en la forma de tratamiento de fallos sino que esta se extiende a su definición. El término fallo presenta distintos matices según el autor que lo emplea, está mejor delimitado cuando se refiere al nivel sensorial [Fernández J. 1998].

La diversidad, proviene de la forma en que se calculan las expectativas y los márgenes de error permitidos. A nivel de sensores se considera que existe un fallo cuando la lectura producida por un sensor no concuerda con las *expectativas* según las especificaciones del sensor. En navegación del

robot, la definición de fallo es más difusa y se utilizan distintos términos (equivocación, error, fallo). En se utiliza el término equivocación (*mistake*) para referirse a eventos en la percepción, conocimiento del navegador o eventos del sistema motriz que desvían al robot de la ruta original. Otros autores consideran que se produce un error cuando una tarea o acción no se finaliza correctamente. Esta última definición puede ser, en algunos casos, demasiado general. Los métodos de diagnosis que se utilizan normalmente en los sistemas de control industrial se aplican a los robots móviles [Fernández J. 1998].

Las soluciones que abordan la detección y recuperación de fallos en los sistemas robóticos móviles están extremadamente ligadas a la arquitectura del software. Por ejemplo, la recuperación de fallos en navegación es raramente usada en sistemas reactivos dado que estos sistemas básicamente reaccionan a eventos. En el contexto de navegación en robots móviles, existen varias arquitecturas que tratan con fallos de diferentes formas y a distintos niveles. En el nivel más bajo, se tratan los fallos de los sensores [Ferrell 1999]. Los procesos de monitoreo y recuperación ante errores son responsables de que el robot ejecute correctamente las tareas encomendadas, detectando cuando esto no es así y ejecutando las acciones de recuperación oportunas. En algunas arquitecturas de tipo deliberativo, el monitoreo se limita a verificar la correcta finalización de cada uno de los pasos trazados en la planificación.

En la detección y diagnóstico de fallos en robots móviles se puede realizar a distintos niveles (hardware, control y supervisión). Ya que las tareas que realiza el robot requieren que funcione bien sin la intervención de un operador humano, y no se efectué ninguna reparación en el tiempo que el robot está en operación.

3.3.1 Influencia de fallos de los sensores y actuadores en el funcionamiento del robot móvil

Los múltiples sensores proveen información confiable y una visión más completa del mundo al robot, un mayor número de actuadores, hace que el robot móvil incremente su capacidad de trabajo. Sin embargo, mayor cantidad de sensores y/o actuadores significan que existen más componentes que puedan fallar, y subsecuentemente degradar el desempeño del sistema robótico. Los fallos físicos pueden ser atribuidos ya sea a fallos mecánicos, fallos electrónicos o fallos sensoriales.

Los cambios sutiles en el estado del robot como puede ser la caída de la señal de un sensor también degradan su desempeño. Las tareas que realizan el robot móvil se basa en su locomoción, ya que la mayoría de las veces las ejecutan sobre terrenos difíciles y peligrosos mientras navegan a través de su entorno, el robot puede darse varias veces golpes con obstáculos, realizar inesperados esfuerzos al subir colinas, o pasar sobre hoyos. Esto provoca deterioros en el hardware del robot. Por lo que no sorprende el fallo en sus componentes o el desajuste a través del tiempo.

La mayoría de los robots experimenta fallo en sus sensores con mayor frecuencia que otros componentes del hardware. Estos fallos tienen una variedad de causas. Por ejemplo, los cables de los sensores se rompen debido a la tensión a la que están sometidos por los puntos de movimiento. Los problemas de montajes de los sensores como puede ser un ángulo insuficiente en la unión de los potenciómetros causando la caída de señal. El valor de referencia de los indicadores de esfuerzo también se cae a través del tiempo. La caja de metal de los potenciómetros en los ángulos de unión, puede no estar bien aislada eléctricamente del resto del sistema lo cual puede causar lecturas erróneas en los sensores. Los actuadores tienen menos fallos que los sensores pero podría decirse que ocupan el segundo lugar en tener fallos en un robot, por ejemplo, el eje del motor tiene tendencia a romperse a través del tiempo. Las abrazaderas del motor se pueden aflojar y produce que el motor rote a un estando no deseado. Mientras el motor rota en su montaje, los cables envían la señal de movimiento al motor se pueden torcer y eventualmente se rompen debido a la tensión.

Los fallos de sensores y actuadores afectan varios niveles de la jerarquía del sistema de control en un robot móvil. El control de bajo nivel es equivalente al nivel sensor-actuador de la arquitectura de control, y el control de alto nivel es equivalente a la locomoción y los movimientos necesarios para que el robot se desplace en su ambiente. Los fallos en sensores y/o en actuadores afectan el control de bajo nivel y consecuentemente dichos fallos pueden causar que el control de alto nivel también se vea afectado, y por consiguiente el comportamiento del robot no resulta ser el adecuado para lograr los objetivos propuestos [Visinsky Monica L 2003].

Por la importancia que tienen los sensores en el sistema de control del robot es común usar de respaldo sensores virtuales que son implementados por software (*soft sensor*), los sensores virtuales utilizan información de los sensores reales, y si existe un fallo en el sensor real puede causar resultados incorrectos en los sensores virtuales. Ya que los sensores virtuales son responsables de activar el correcto comportamiento del robot cuando ocurre un fallo en un sensor real. La utilización de sensores virtuales en los robot es una buena técnica de redundancia sin embargo aumenta el código de programación.

Es primordial detectar y confirmar los fallos en el nivel más bajo. Si embargo el fallo puede no ser confirmado en nivel en cual se originó, entonces los niveles superiores lo deberán detectar y compensar. Mientras el fallo se propague hacia los niveles superiores de la jerarquía en sistema robótico, éste se propagará y afectará el funcionamiento del sistema en mayor grado. Si el tiempo de respuesta es mayor para detectar un fallo significará que las manifestaciones pueden propagarse a otros niveles. Por lo que detectar y confirmar los fallos en los niveles más bajos de la jerarquía del sistema robótico, maximiza la efectividad de los procedimientos de recuperación y minimiza el impacto del fallo en el desempeño de todo robot. Otros tipos de fallos pueden ocurrir, ya sea en el sistema de procesamiento o de software, pero estos fallos ocurren con muy poca frecuencia [Alanis G. A., Ors Carot R.., Serrano J.J. 2004].

3.4. Robots con implementación de mecanismos tolerantes a fallos

3.4.1 El robot Hannibal

La tolerancia a fallos en el robot Hannibal fue implementada por Cinthya Ferrell en los laboratorios de Inteligencia Artificial del MIT, la implementó utilizando una red distribuida de procesos concurrentes. Para tolerar los fallos del *Hardware*, Ella diseñó una serie de procesos que toleran los fallos en cada componente. Estos procesos son responsables de detectar fallos de sus respectivos componentes, y minimizar el impacto del fallo en el desempeño del Robot. Explota la concurrencia de procesos, y utiliza la distribución de sensores y actuadores en el robot, los monitores están diseñados para detectar y compensar los fallos para cada componente simultáneamente.

3.4.1.1 Replicación de hardware

El hardware se duplica comúnmente para incrementar su confiabilidad. En Hannibal, algunos sensores y actuadores son replicados utilizando técnicas adecuadas para no enmascarar el fallo. Por ejemplo, utiliza múltiples potenciómetros para captar el ángulo de unión, se utiliza un árbitro que reúne los valores de los ángulos de unión de cada potenciómetro y establece el valor aceptado del ángulo de unión para que sea el valor más común. La aplicación de software utiliza este valor como punto de unión. El fallo de algún sensor del ángulo o punto de unión se detecta si su valor no concuerda con el

obtenido por los otros sensores. Sin embargo, el fallo se enmascara debido a que el valor más común es tomado como valor real. Asumiendo que la mayoría de los potenciómetros trabaja correctamente, la recuperación de un fallo es inmediata ya que el software proporciona el valor correcto del ángulo de unión. Consecuentemente, los fallos de hardware son detectados y confirmados limitados a nivel de hardware de la jerarquía del sistema.

Lo ideal es detectar y corregir los fallos de hardware en ese nivel. Sin embargo, esto tiene algunos inconvenientes primero, la replicación de los sensores y actuadores es costosa.

3.4.1.2 Redundancia de comportamientos

La implementación de una estrategia redundante de comportamientos tolera fallos en el control de alto nivel. Al utilizar esta estrategia, el controlador fue diseñó integrando comportamientos redundantes para realizar una determinada tarea. Existe un modelo de desempeño para cada estrategia, y el fallo es detectado si el desempeño del comportamiento actual es peor que el desempeño esperado. Si la primera estrategia que se intenta no es suficiente, el controlador eventualmente intenta otra estrategia. El controlador busca en un repertorio de estrategias hasta que encuentra una estrategia con un desempeño aceptado en lugar de estar tratando sin éxito la misma rutina.

Hannibal tienen comportamientos redundantes en su caminar cada comportamiento se implementa con un modo de andar diferente (algunos de estos modos de andar son inestables). Si una pata le falla busca un modo de andar que pueda sobrellevar la locomoción inestable hasta que el controlador encuentre un modo de andar que sea estable a pesar de haber perdido la pata. Hannibal no puede reconocer que pata falló y adaptar su modo de andar para asignar específicamente el fallo. La estrategia de redundancia requiere inherentemente que los fallos de hardware se manifiesten en el comportamiento del robot antes de que el sistema de control pueda detectar que algo esta mal. Por esta razón, el desempeño de los comportamientos infectados debe degenerarse a un nivel inaceptable antes de que el controlador tome acciones correctivas. Esto es perjudicial para un robot que debe funcionar en ambientes peligrosos. Un ejemplo es el caso de que un sensor se dañe entonces el desempeño del su sensor virtual tendría que degradarse suficientemente antes de que el sistema de control se percate del fallo. La disponibilidad de Hannibal depende de detectar, enmascarar y recuperarse de fallos a bajo nivel del sistema de control antes de que los estos infecten a los comportamientos de más alto nivel.

Otro ejemplo que utilizó redundancia de comportamientos fue en un submarino autónomo cuando trata de evitar el fondo del océano. Existen varias estrategias que el submarino puede utilizar para evitar el fondo: Subir utilizando aletas, subir utilizando el balasto frontal, incrementar su ligereza para mantenerse a su nivel, y utilizar un sostén para alejarse del fondo. La estrategia preferida es la de impulsar el vehículo hacia arriba usando las aletas. Si el actuador de la aleta esta roto o no funciona el submarino no podrá evitar adecuadamente el fondo, sin embargo, con los comportamientos redundantes se puede recupera cambiando la estrategia de emerger con el balasto frontal. Si el desempeño es insatisfactorio todavía, el controlador intenta otras estrategias hasta que el desempeño sea aceptable [Ferrell Cinthya 1999].

3.4.1.3 Sensores virtuales robustos

El controlador de Hannibal utiliza sensores virtuales robustos para confirmar los fallos de hardware a bajo nivel. Los sensores virtuales se mantienen confiables a pesar de la aparición de fallos en los sensores reales, ya que son responsables de caracterizar la interacción del robot con su ambiente utilizando la información de los sensores reales y de activar los comportamientos del robot.

Si los sensores virtuales arrojan la información correcta a pesar de presentar fallos en los sensores reales, entonces el robot continuará funcionando correctamente a pesar de dichos fallos. Por ejemplo, si el sensor virtual (de contacto con el suelo) puede determinar correctamente el contacto con el suelo a pesar de que el sensor de avance presenta un fallo, los comportamientos que utilizan la información del sensor de contacto con el suelo no son afectados debido al fallo. Los sensores virtuales están siendo utilizados porque limitan el efecto de los fallos en el control de bajo nivel y previenen que los fallos infecten al control de alto nivel. Esto compensa efectivamente los fallos locales. Los fallos locales (también llamados fallos no catastróficos) son fallos cuyo efecto esta limitado a la pata donde puede ocurrir. Por ejemplo, el fallo del sensor localizado en el tobillo de una pata es un fallo local porque afecta la habilidad de la pata para sentir el contacto con el suelo, pero no afecta la habilidad de las otras patas para sentir el contacto con el suelo.

Algunos fallos afectan el comportamiento de todo el sistema y se denominan fallos globales. Los fallos globales (fallos catastróficos) deben ser compensados en el nivel alto. Por ejemplo, si el actuador del soporte de la pata se daña entonces la pierna no podrá soportar el cuerpo, consecuentemente, este fallo afecta la estabilidad global del robot. El control de alto nivel deberá compensar dicho fallo cambiando el modo de andar del robot para que este pueda caminar de manera estable con una pata menos.

Entonces se puede decir que Hannibal compensa los fallos locales dentro del control de nivel más bajo, y compensa los fallos globales en el control del nivel más alto. Por ejemplo, se diseñaron tres sensores virtuales redundantes para detectar el contacto con el suelo, estos utilizan diferentes sensores reales para percibir el contacto con el suelo. Cada sensor virtual redundante vota falso o verdadero, y el veredicto es determinado por el voto de mayoría.

Las capacidades de tolerancia a fallos en Hannibal son implementadas utilizando agentes adaptativos o ajustables.

.

No se utilizó redundancia en los sensores virtuales robustos, por las siguieres razones: Primero, no hay manera de que el controlador pueda obtener un veredicto confiable una vez que la mayoría de los sensores falle. Segundo, el tamaño del código crece significativamente con esto ya que múltiples sensores virtuales son codificados para cada interacción pata-terreno que se deseaba determinar. Dada a la pequeña memoria de Hannibal esto es una consideración muy importante. Tercero, una vez que un sensor real falla, todos los sensores virtuales que utilizan información sobre este sensor se vuelven menos confiables, por lo tanto, una minoría de fallos en los sensores reales podría afectar a la mayoría de los sensores virtuales redundantes. [Ferrell Cinthya 1999] Presenta un ejemplo donde todos los sensores virtuales que activan los comportamientos para que el submarino autónomo evite el fondo del mar, utilizan un sensor real de altitud, si este sensor falla, entonces la confiabilidad de todos estos sensores virtuales se degeneran. Consecuentemente, el desempeño de *evitar el fondo* se degrada con el fallo de un solo sensor real. La confiabilidad en los sensores virtuales redundantes que reciben información de un solo sensor real no es muy adecuada en cierto tipo de fallos. Los sensores virtuales que utilizan información de diferentes sensores reales generalmente producen un resultado más confiable.

Hannibal utiliza sensores virtuales robustos basados en la adaptación en lugar de la redundancia.

Ejemplo de adaptabilidad utilizada en fallos globales en el robot Hannibal; cuando una pata sufre un fallo y deja de ser útil al robot. El control de alto nivel debe cambiar el modo de andar de tal manera que la locomoción debe de permanecer estable con una pata menos.

La implementación de los comportamientos redundantes en el andar de Hannibal requiere una gran cantidad de código, ya que cada comportamiento exhibe una manera de caminar diferente además del mecanismo de cambio de manera de andar, esto se realiza para las 6 patas que componen el sistema de locomoción del robot. En contraste, el modo adaptable implementa un comportamiento de manera de andar que este se puede alterar al cambiar un solo parámetro. El control de bajo nivel es responsable de detectar fallos globales y de alertar al control de alto nivel. El control de alto nivel es responsable de adaptar el comportamiento del robot de tal manera que la locomoción permanezca estable.

3.4.1.4 Restricción de fallos

El sistema previene efectivamente que los fallos de los sensores y actuadores para que no influyan adversamente en el comportamiento del robot. Para cumplir esto, los métodos de detección monitorean las salidas del sensor y alertan al sistema de un posible fallo si llega a surgir. De tal manera que el sistema puede compensarse efectivamente los fallos una vez que sabe cuando y que tipo de fallo ha ocurrido. Por ejemplo, los sensores virtuales robustos filtran los efectos de los fallos para que estos no influyan en el comportamiento del robot. El sistema detecta y se recupera exitosamente antes que el desempeño del robot se degrade hasta llegar a un nivel inaceptable. La respuesta rápida que se implementó dio como resultado un éxito en la restricción de fallos.

Los procesos de recuperación mantienen el desempeño en el nivel alto dado el estado funcional del hardware. Ya que el sistema reconoce los fallos, los procesos de recuperación se adaptan específicamente con el uso de sensores y actuadores para minimizar los efectos de los fallos en el comportamiento del robot. A nivel del sensor virtual, los procesos de recuperación intercambian velocidad por confiabilidad mientras el número de sensores funcionales fue decrementado. A nivel de locomoción, los procesos de recuperación intercambian velocidad por estabilidad mientras el número de patas utilizables va en decremento. Los procesos de reintegración son los que reincorporan los componentes reparados de tal manera que el robot tenga acceso a todos sus recursos de manera confiable. El tiempo de respuesta de la reintegración es relativamente rápido de tal manera que el sistema no tiene que esperar mucho tiempo antes de que pueda reutilizar sus componentes reparados.

En Hannibal se implementaron todas las capacidades de la tolerancia a fallos dentro del software. Y no se implementaron las capacidades de la tolerancia a fallos a nivel de hardware por consideración de peso y costo. Sin embargo el tamaño del código se incremento significativamente, de tal manera que los procesos tolerantes a fallo requirieron la misma cantidad de código, que el que se requirió para lograr su locomoción, y poder transitar terreno áspero (ambos).

Aplicando las técnicas que fueron implementadas en el robot Hannibal pudo su diseñadora soportar como máximo 54 fallos (48 fallos no-catastróficos, 3 fallos catastróficos en la pata central izquierda, y 3 fallos catastróficos en la pata central derecha) antes de que no pueda caminar de una manera estable. Por lo tanto, es posible que el robot pueda trabajar con la minoría de sus sensores activos.

En Hannibal los fallos tomados en cuenta para ser tolerarlos son solamente los los producidos en sensores y actuadores, y son tratados en el nivel más bajo posible para que las capas superiores no se enteren de tales fallos.

3.4.2 El robot Xavier

Xavier es un robot diseñado en el laboratorio (*Robot Learning Laboratory)* de la Universidad de Carnegie Mellon, Pittsburg (USA). Y sirvió para la implementación mecanismos tolerantes a fallos a

nivel de navegación por Joaquín López Fernández del Departamento de Ingeniería de Sistemas y Automática de la Universidad de Vigo en España.

3.4.2.1 Situación de fallo y situación de excepción

Una característica peculiar que tiene un robot cuando realiza las tareas de navegación en un entorno dinámico e interactivo, que se puede encontrar en demasiadas y diferentes situaciones anómalas, que es difícil tomarlas todas en cuenta para realizar su detección y diagnostico de fallos. Por lo que se les clasificó en:

Situación de fallo pudiendo tener fallos en algún dispositivo o fallos en el sistema de navegación.

Situaciones no contempladas en el desarrollo del sistema de navegación que no pueden ser consideradas como fallos. El sistema funciona de acuerdo a las expectativas, pero el problema surge porque en la elaboración del sistema de navegación no se han contemplado éstas situaciones [Fernández L. J. 2002].

Ejemplo, Suponga que un robot móvil que realiza sus tareas en el interior de un edificio (vigilante en un banco, llevar la comida a los enfermos en sus camas en un hospital) utiliza un sistema de navegación jerárquico con dos capas. En la capa superior un planificador calcula la trayectoria a seguir dados los puntos origen y destino. Dicha planificación la realiza partiendo de un plano del edificio en el que se mueve. En el nivel inferior existe un módulo reactivo que se encarga de seguir luna determinada trayectoria determinada por el planificador, evitando obstáculos que encuentra en el camino. La información entre los módulos se produce en una sola dirección. Solo existe flujo de información del nivel de planificación al nivel reactivo. Mientras el robot trata de seguir una trayectoria a lo largo de un pasillo se pueden producir diferentes situaciones:

Situación 1: Puede ser que el robot no sea capaz de seguir la trayectoria porque un fallo en los sensores le indican que hay un objeto en el camino y pretenda evitarlo por lo cual no podrá avanzar.

Situación 2: Puede ser que el pasillo esté realmente bloqueado y el nivel reactivo intente bordearlo. Al encontrarse en un pasillo cerrado no saldrá de esta situación a menos que el planificador le indique un camino alterno, pero como el flujo de información se realiza en una única dirección, la trayectoria será siempre la misma.

En la primera situación existe un fallo debido a que un componente no funciona de acuerdo a las especificaciones, en la segunda situación no se puede decir que haya un fallo aunque el sistema no responda según lo esperado. El sistema de navegación en esta situación funciona de forma correcta, al igual que el sistema sensorial y demás partes del robot. El problema, del segundo caso, es que el robot se ha encontrado en una situación que no se había tenido en cuenta en la etapa de diseño .

Por lo que [Fernández L. J. 2002] se refiere a *excepciones o situaciones de excepción* tanto a situaciones de fallo como situaciones no contempladas. Para que el robot opere de forma continua y autónomamente y se deben de tener en cuenta ambas. Debido a esta clasificación de situaciones en las que se puede encontrar el robot El sistema de detección y diagnóstico de fallos en el trabajo realizado con Xavier, se considera como un sistema de detección y diagnóstico de excepciones.

Por lo que se consideró importante dividir el espacio de estados del robot en dos subespacios:

El subespacio de *excepciones,* que engloba aquellas situaciones en las cuales el robot no llegará a terminar la tarea encomendada o bien no lo hará en el tiempo requerido. Subespacio *nominal,* que

incluye el resto de las situaciones en las cuales el robot, de seguir así, realizará la tarea con éxito (figura 3.9).

Algunos de los estados de excepción (representados por cuadrados vacíos) son conocidos de antemano, mientras tanto existen otras situaciones (representados por cuadrados sombreados) que son totalmente desconocidas. Los estados nominales se representan con círculos.

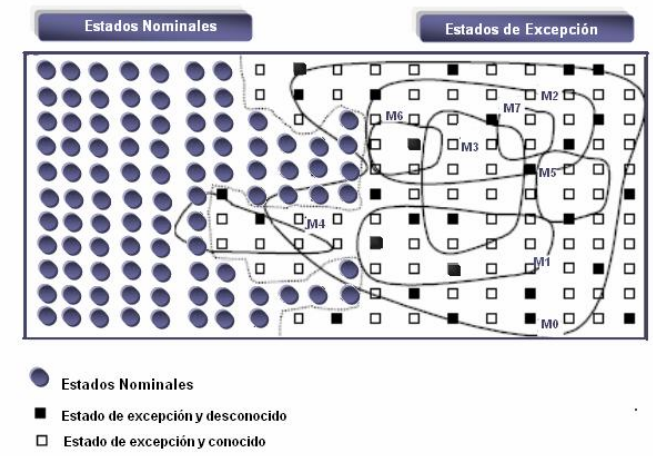


Figura 3.9 División del espacio de estados del robot.

A pesar de que conceptualmente se realizó la separación entre ambos subespacios y están bien definidos, en la práctica no es una tarea fácil realizar, el lograr que un sistema detecte tales situaciones por diversas razones:

a) La información de que dispone el robot a través de los sensores es una información parcial de la realidad.
b) Los datos de los sensores presentan ruido asociado a la información.
c) Dependiendo de las condiciones del entorno, las lecturas que en un momento indican una situación de bloqueo, en otro instante puede no serlo.
d) Puede haber un número muy grande de excepciones, la mayoría de las cuales son desconocidas en el momento de diseño del supervisor.

3.4.2.2 Modelos de tolerancia a fallos implementados en el robot Xavier

La mayoría de detección de fallos se ha enfocado en la utilización de generación de residuos, argumentando que la parte de decisión basada en residuos bien diseñados es relativamente fácil [Chen J., Patton R.J., Chen Z. 1998]. Sin embargo, resulta muy difícil modelar matemáticamente el comportamiento global del robot como se hace en otros sistemas, pero sí se puede modelar el funcionamiento de algunos dispositivos o módulos para detectar fallos locales.

Debido a lo anterior la implementación de la tolerancia a fallos en el robot Xavier se le dio un nuevo enfoque, al diseñar *dos modelos para detectar y diagnosticar fallos*:

1.- Modelo determinista es aquel que implementa una nueva arquitectura soportada por un conjunto de monitores organizados jerárquicamente que obtienen información acerca del funcionamiento del robot mientras realiza las tareas de navegación (evitar obstáculo, planeación de trayectorias, estimación de la posición del robot, estimación de la posición del objetivo del robot, etc.). Un monitor es una tarea continua en tiempo real que supervisa el comportamiento del robot buscando síntomas o indicios de

fallo en el robot. Un síntoma es una manifestación de malas prestaciones o valores de los residuos fuera de los márgenes permitidos.

Los monitores están organizados en dos niveles (superior e inferior) teniendo como objetivo proveer una amplia cobertura para situaciones de fallo, incluyendo aquellas que no pueden ser gestionadas de forma estándar por el sistema de navegación.

En la capa superior de la arquitectura están situados los monitores generales que detectan un gran número de fallos, los situados en las capas inferiores son monitores más específicos que detectan situaciones concretas. Los monitores de la capa superior por ser generales son más lentos en la detección, en comparación a con los específicos que son mucho más rápidos, ya que están enfocados a un problema concreto. La ventaja de esta arquitectura en su fase diseño es que se puede inicial el diseño con los monitores generales, y a medida que se va obteniendo experiencia e información acerca de los problemas específicos, se van diseñando los monitores necesarios.

El sistema supervisor posee tiempos de respuesta muy cortos, debido a que la fase de detección y diagnóstico es muy sencilla. La fase de detección utiliza una serie de monitores en el que cada uno constituye un sistema de extracción de síntomas.

2.-Modelo Estocástico la necesidad de tratar con la incertidumbre de la información de entrada al sistema de supervisión. Esto es debido, en gran medida, y la dificultad de encontrar monitores (síntomas) que se disparen (identifiquen) únicamente ante situaciones de excepción, y en ningún caso, ante situaciones normales.

Este modelo trata de optimizar el proceso de decisión asumiendo que los síntomas no son completamente fiables (trabajando así con probabilidades). Este modelo, al igual que modelo determinista, está basado en mensajes de los monitores, sin embargo en este caso se toma en cuenta la incertidumbre acerca de su relación con una situación de excepción, la posibilidad de fallo de las acciones y el costo de las mismas. El objetivo es que el supervisor tome las decisiones óptimas conociendo de antemano las probabilidades de fallo de las observaciones, y de las acciones. Se considera una decisión óptima cuando, se tiene la información disponible, y la decisión tomada conlleva una bonificación máxima a largo plazo. El sistema de bonificación es definido por el diseñador, quién decide qué tareas son más importantes o qué recursos son más costosos.
.

Este modelo además de tener en cuenta el costo de las acciones incluye el tratamiento de la incertidumbre a distintos niveles:

- *Incertidumbre en las observaciones*
- *Incertidumbre en el resultado de las acciones.*

La detección de fallos en la navegación de robots móviles y su recuperación es completamente diferente tanto al tratar la incertidumbre acerca de la validez de la información proveniente de los observadores, como en la toma de decisiones.

En esta aproximación utiliza el modelo de Markov parcialmente observable (POMDP) con ciertas modificaciones. Resumiendo, las principales características del sistema son:

- El proceso es descrito usando un conjunto de estados finitos.
- El sistema supervisor puede actuar sobre el proceso a través de un conjunto de acciones.

- La dinámica del sistema se describe mediante un esquema de probabilidades de transición de estados.
- El supervisor posee información sobre el estado de proceso a través de un conjunto finito de observadores.
- La calidad de la supervisión viene dada por medio de un sistema de bonificación o costo asociados a estados o transiciones.
- El objetivo del supervisor es maximizar una función que combina los costos y las bonificaciones obtenidas a lo largo del tiempo.

En la figura 3.10 se presenta la división de los problemas con los que Xavier se puede encontrar [Fernández J. 2002]. Existen multitud de componentes que pueden fallar, e incluso puede ser que una parte del sistema de control, tal como el módulo reactivo, en ciertas situaciones no se comporte como debería.

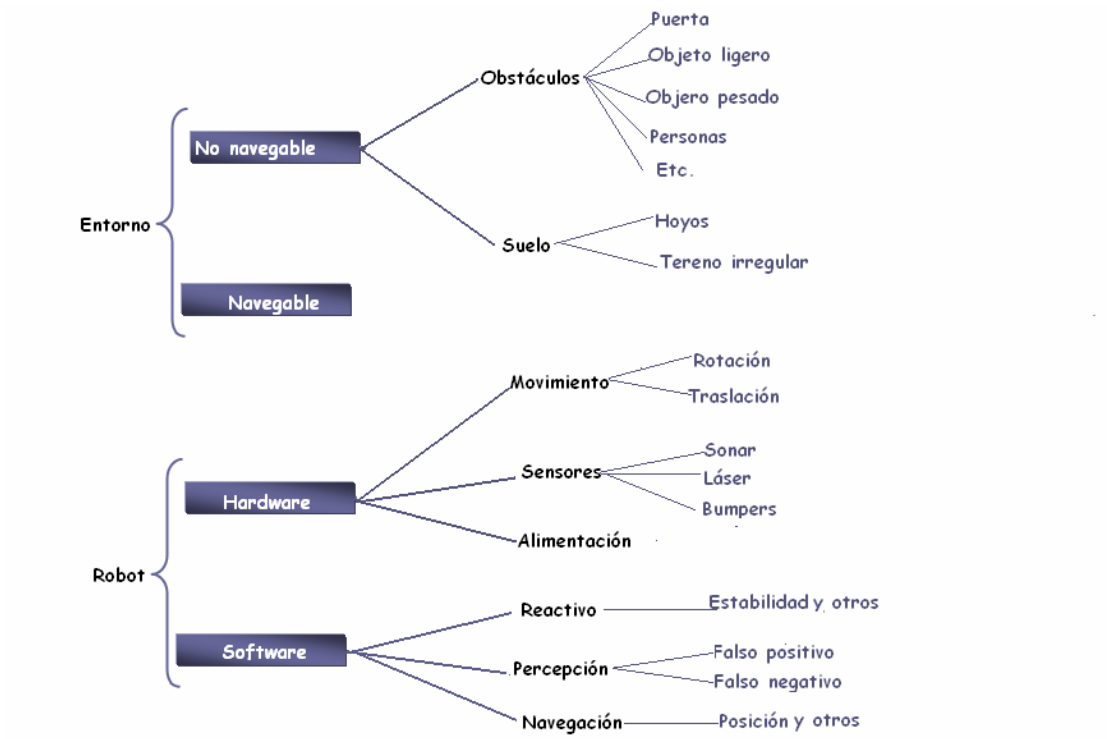


Figura 3.10 Clasificación de posibles problemas de navegación que se le pueden presentara un robot móvil.

En el modelo propuesto se aprovechan los monitores que fueron desarrollados para implementar el modelo determinista y se añaden otros métodos de obtener información acerca del estado de Xavier.

En éste modelo desarrolla una arquitectura genérica, puede se aplicada en otro tipo de problemas similares con solo cambiar el modelo (observaciones, acciones, estados) junto con sus parámetros (estadísticos que caracterizan el modelo). Los parámetros del modelo se seleccionaron basándose en la experiencia, en otros casos que pueden ser aprendidos con métodos de aprendizaje automático tales como aprendizaje por refuerzo. El sistema de supervisión (figura 3.11) es válido para todos los problemas que se puedan modelar mediante el proceso de decisión de Markov parcialmente observables (POMDP) simplificado. No hay más que sustituir la definición del modelo por la del nuevo proceso.

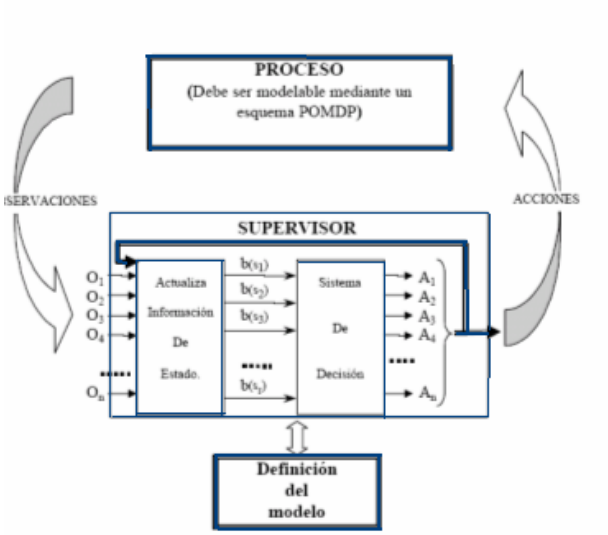


Figura 3.11 Sistema para cualquier proceso que sea posible modelar como un POMDP.

3.4.3 Robot soldador

En Robot soldador diseñado por la Universidad de Texas en el departamento de robótica y automática, tiene implementado un software de monitoreo y detección de fallos a partir de la adquisición de las señales de mando y velocidad de cada uno de los motores correspondientes a los ejes o grados de libertad integrados en sus articulaciones.

Para su solución se desarrolló el software gráfico denominado *LabVIEW* para control automático en tiempo real, en el que se puede crear aplicaciones determinísticas avanzadas con más de 500 rutinas de análisis y control, permite la adquisición y almacenamiento de señales analógicas y digitales , así como la generación de un historial de las mismas. Tiene integrado un conjunto de herramientas que permiten el análisis offline de los datos recolectados, tiene integrada de una red neuronal artificial capaz de generar la señal necesaria para una determinada velocidad de referencia en la situación de ausencia de fallo en el sistema. El software además permite la generación de reportes con la información más relevante de cada toma de datos.

El sistema es capaz de representar las variables de funcionamiento, velocidad y corriente eléctrica de mando, de los motores de los robots incorporando funciones especiales. Además el sistema calcula los parámetros (corriente eficaz, velocidad media, coeficiente de disipación, número de cambios de sentido etc.) que permiten el diagnóstico de los motores que tienen integrados los robots.

El software genera una base de datos mediante la creación de hojas Excel, en función del número de máquinas, robots y algunos otros datos adicionales que puedan ser de interés, haciendo uso de las herramientas Active X, que permiten el control total de Excel desde el propio *LabVIEW*. El programa incorpora además un sistema de diagnóstico basado en una red neuronal artificial. La red se ha integrado dentro del propio *LabVIEW*. La detección se realiza sobre los motores que permiten el movimiento de cada uno de los seis ejes de los que disponen los robots. Para ello se miden las señales que proporcionan las variables que controlan cada uno de los motores. Los motores de los robots son síncronos, trifásicos, de magnetización permanente, y están equipados con un resolvedor que permite el cálculo por un lado de la velocidad de giro del motor, así como de su posición. En estos motores existen dos lazos de control. Un lazo de control externo de velocidad y uno interno de corriente.

Las señales que proporcionan las variables son por un lado la corriente de mando, que sirve como referencia para conseguir el sistema trifásico equilibrado de corrientes que alimentan el motor y por otro la velocidad de giro de dichos motores.

De forma general la idea fue conseguir determinar la relación existente entre la velocidad de giro y la corriente de mando de cada uno de los motores, y la corriente de mando de cada uno de las variables. Ante la dificultad que implica la elaboración de un modelo analítico que representa dicha relación, y teniendo en cuenta los datos de que se dispone, se optó por la identificación de esta relación mediante la utilización de un modelo neuronal, que va implementado dentro del propio LabVIEW, del tipo NARX.

El software permite la adquisición, visualización y almacenamiento de los datos en un entorno gráfico guiado por menús. Para la realización de estos menús se utilizaron algunas de las posibilidades que presenta la Picture Control Toolkit. A continuación se analiza cada módulo:

La opción de adquisición de los datos permite seleccionar al usuario, por un lado las características correspondientes al robot (tipo, número de robot, número de línea y taller al que pertenece) , que le servirán para almacenar los valores recolectados y generar las hojas Excel correspondientes, y por otro lado los parámetros propios de una adquisición de datos, como pueden ser los distintos canales de la tarjeta de adquisición de datos que van a ser utilizadas, frecuencia de muestreo, así como el inicio y la duración de dicho muestreo. Una vez finalizada la adquisición de los datos, el usuario puede dentro esta opción comprobar en ese instante la forma y valores de las señales recogidas para cada uno de los ejes del robot.

El núcleo central del software permite el estudio de los datos recogidos así como la detección de posibles fallos en el sistema es la parte del módulo denominada *Análisis de los datos recogidos*

El módulo del historial, ofrece al usuario la posibilidad de recuperar los valores, así como observar las formas de onda de cada una de las señales que se han recogido y que forman parte del historial de datos, mediante búsquedas por tipo de robot. Además es posible obtener la forma de onda del ángulo girado, obtenido por integración de la velocidad, y conocer el valor máximo de las señales, y observar en forma gráfica el punto donde el máximo se ha producido.

Dentro del módulo *Diagnóstico* se ofrece de forma tabulada el valor de los parámetros más significativos para cada uno de los ejes del robot. Estos parámetros son: Intensidad máxima, Intensidad eficaz, Velocidad máxima, Velocidad media, Número de cambios de sentido, Coeficiente de disipación (tiempo durante el cuál se mueve el motor frente al tiempo total del ciclo) y Desplazamiento máximo. El análisis de estos datos a lo largo del tiempo permite estudiar su evolución de forma que se pueda también realizar la detección de fallos a partir del análisis de tendencias mediante el fijar valores de umbrales. Dentro de esta parte está incluido el módulo de *detección de fallos mediante una red neuronal artificial*.

El software presenta en forma gráfica la señal de respuesta de la red neuronal artificial, que corresponde a una situación sin fallo del sistema para la señal de velocidad correspondiente, frente a la señal real medida con éste sistema. De la diferencia de ambas señales será capa de detectar los funcionamientos anómalos.

También ofrece de forma gráfica el error o diferencia entre ambas señales indicando las zonas donde se ha sobrepasado el límite permitido. El objetivo de esta herramienta es poder detectar los picos de sobreintensidad que se producen y que afectan de forma negativa a la vida de los variables.

El módulo de *Análisis* ofrece la posibilidad de comparar de forma gráfica las señales almacenadas con anterioridad, de tal manera que se pueda visualizar rápidamente si se ha producido algún cambio significativo en las señales con respecto al tiempo.

Por último el módulo de *Tendencias* muestra gráficamente la evolución de los valores de corriente máxima y eficaz, así como los valores de velocidad máxima y media para las diferentes tomas de datos realizados a lo largo del tiempo.

El software tiene la posibilidad de generar informes. Para esto se hace uso de las herramientas de *Report Generation* que se incluyen en la versión de LabVIEW 6.0, permitiendo la generación de informes en formato html. Dentro del informe se incluyen los datos característicos del robot, así como los parámetros anteriormente indicados y las gráficas de corriente y velocidad correspondientes a cada uno de los ejes. Por otro lado se generan además un conjunto de hojas de Excel para cada una de la toma de datos realizados donde se recogen además de los datos característicos del robot, y los valores de los parámetros característicos.

Después de los descrito en éste capitulo tolerancia a fallos implementada en el diseño de los sistemas robóticos, se ha realizado de una manera parcial, y aunque se duplican ciertos componentes de hardware en este tipo de sistemas es imposible realízalo a grande escalas ya que su costo, tamaño y peso no son permitidos para un buen funcionamiento, sin embargo los diseñadores de robot implementan mecanismos tolerantes a fallos por medio de software, ya sea por medio de monitores como es el caso de Xavier o de sensores virtuales en Hannibal, ya que esto tiene un menor costo de implementación, implicando de cierta manera que sea complicado, ya que se basa en modelos matemáticos complejos y difícil de implementa.

3.5 Conclusiones del capítulo

La inteligencia artificial ha encontrado en el dominio de los robots móviles una plataforma idónea para el desarrollo y verificación de algoritmos relacionados con temas de planificación, aprendizaje y razonamiento.

Su entorno casi nunca es modelable ni predecible y las acciones tampoco son deterministas. Por lo que los diseñadores de estos robots se han visto en la necesidad de enfocarse a desarrollar primordialmente algoritmos que puedan hacer más eficiente estas tareas y investigo poco acerca de métodos eficientes para tolerar fallos en el hardware y/o software en estos sistemas, sin embargo en los últimos años debido a limitaciones de los sensores, donde la mayor parte de la información adquirida a través de estos es incompleta e imprecisa ha producido constantemente fallos en el sistema de navegación, de la misma manera, los requerimientos actuales de los robots a que permanezcan el mayor tiempo disponibles debido a el tipo de tareas que deben de desempeñar, se ha tenido que tomar en cuenta la implementación de técnicas tolerantes a fallos en este tipo de problemas.

La detección y diagnóstico de fallos en robótica móvil requiere del uso de una metodología, como se hace en los sistemas de control industriales, de tal manera que se garantice que el sistema de control a implementar sea fiable y seguro.

Como se puede observar por lo anteriormente expuesto sobre este tema el tratamiento de fallos en robótica móvil esta ligado a realizar obtención de síntomas, detección, y diagnóstico de fallos, y no deben estar tan aisladas como se hace en los sistemas industriales.

A pesar de los esfuerzos de los diseñadores de implementar mecanismos tolerantes a fallos no han logrado realizarlo de una manera integral, diseñando módulos de detección y diagnóstico de fallos que sean fácilmente implementados, adaptables y flexibles, de tal manera que no interfieran con la arquitectura software, hardware y el sistema de control del robot sino que sirvan de su complemento y así garantizar el cumplimiento efectivo de las tareas que el robot debe de realizar.

Por último se puede enfatizar que los robots móviles son las máquinas que en los próximos años acompañaran al hombre a realizar diversas tareas complejas, por lo que las nuevas generaciones demandarán estos sistemas e inevitablemente se diseñarán con mayor grado de complejidad, lo que ocasionará que sean difíciles de controlar, y será necesario desarrollar arquitecturas tolerantes a fallos robustas, flexibles y adaptativas, de tal manera que los robots satisfagan adecuadamente las tareas para lo que se diseñan y operen con mayor disponibilidad.

Capítulo 4 Análisis del sistema tolerante a fallos

4.1 Introducción

En los últimos años, las nuevas aplicaciones que se le han dado a los robots han incrementado la importancia de tolerar los fallos. Ya que los robots se están utilizando actualmente para realizar tareas en ambientes peligrosos, tales como en áreas radioactivas donde no es muy fácil darles mantenimiento regularmente por lo que es necesario se mantenga operando por largos periodos de tiempo. También se están aplicando en situaciones donde se requiere realicen tareas extremadamente refinadas y con alta precisión como es el caso de los robots ayudantes en cirugías médicas de alto riesgo, los fallos en este tipo de trabajos pueden ocasionar una catástrofe. Además esta tecnología ha tenido un gran repunte, actualmente se han diseñado una gran cantidad de robots para realizar trabajos en complejas líneas de manufactura donde los ambientes no son estructurados, en estos espacios de trabajo tan impredecibles para la movilidad de los robots, los fallos son muy comunes y es difícil anticiparlos cuando se realiza su diseño.

La arquitectura tolerante a fallos mediante un Sistema Multiagente propuesta en este trabajo, es un sistema computacional complejo por lo que es necesario realizar el análisis del funcionamiento del sistema de control del robot móvil (para este trabajo se utilizó el robot móvil descrito en el capítulo tres utilizando la arquitectura software 3+ modificada) con el objetivo de poder diseñar adecuadamente:

- La arquitectura de control del robot que soportará al SMA tolerante a fallos.
- Los diferentes tipos de agentes que constituyen al SMA.
- Las comunicaciones necesarias entre los agentes.
- Las tareas que realizarán cada agente para tolerar los fallos en los dispositivos físicos y tareas que constituyen el sistema de control del robot.

De tal manera que los agentes que integrarán el SMA ejecuten eficientemente los mecanismos tolerantes a fallos sobre el sistema de control del robot, obteniendo con esto mayor tiempo de funcionamiento del robot.

4.2 Diseño modificado del sistema de control distribuido para un robot móvil

Actualmente las arquitecturas de control distribuidas (a nivel físico) son una de las soluciones utilizadas en los nuevos diseños del sistema de control en los robots móviles, esto se debe al aumento en la complejidad del *hardware*, así como a la interacción entre manifestaciones inteligentes y las restricciones de tiempo real. Este tipo de arquitectura están constituidas generalmente una serie de nodos interconectados entre por un Bus industrial que puede ser: CAN, ProfiBus o Device Net. El diseño la arquitectura física para el sistema de control en un robot móvil se debe de realizar de tal manera que se pueda adaptar a las necesidades de la aplicación del robot. Una configuración genérica de éste tipo puede estar constituida por un microprocesador en cada nodo, dedicado a procesar la información de los sensores y actuadores conectados a dicho nodo, así como las tareas específicas que debe realizar el robot para ejecutar el trabajo para lo cual fue diseñado.

En nuestro caso particular se decidió por un sistema de control distribuido al que se le realizaron algunas modificaciones con la finalidad de que la arquitectura tolerante a fallos mediante el SMA pueda trabajar de una manera más eficiente.

A continuación se describe el diseño de la arquitectura de control distribuida modificada para un robot móvil que nos permitirá soportar eficientemente la arquitectura tolerante a fallos. Éste diseño se realiza con la finalidad de poder duplicar algunos dispositivos de control críticos, así cómo poder realizar una doble conexión de todos los sensores y actuadores con la finalidad permitir la funcionamiento los dispositivos en otro nodo en caso que el nodo donde se localicen conectados falle totalmente.

4.2.1 Clasificación de los dispositivos de entrada y salida que integran el sistema de control del robot

Lo primero que se realizó para iniciar el análisis del sistema de control del robot fue la clasificación los diferentes tipos de dispositivos de entrada (sensores) y de salida (actuadores) conectados en los nodos que integran la estructura física del sistema de control (Tabla 4.1). Estos dispositivos se clasificaron dependiendo del número que se requiere conectar a cada nodo:

Tipo de dispositivo	*Descripción*
Dispositivo múltiple	***Dispositivos de entrada y salida que son del mismo tipo; se conectan de manera múltiple en un solo nodo para abarcar un mayor rango de acción o una mayor precisión ejemplo, anillos de sensores infrarrojos, anillos de bumpers, anillos ultrasónicos, etc.***
Dispositivo único	***Dispositivos de entrada y salida que se conectan de manera única a un determinado nodo, ejemplo, batería, cámara, sensor de control de batería, sensor de temperatura, etc.***

Tabla 4. 1 Tipos de dispositivos múltiples o únicos y su descripción.

4.2.2 Distribución y tipo de conexión de los dispositivos en los nodos que integran el sistema de control

Cada nodo que integra el sistema de control del robot móvil lo constituyen físicamente:

- Sensores.
- Actuadores.
- Microcontrolador.
- Controlador de la red.
- Conexiones físicas de los dispositivos
- Memorias.

Cada nodo tiene conectados solamente en estado activo dos dispositivos de entrada y/o salida, y dos dispositivos de entrada y/o salida en estado dormido en doble conexión como máximo. La figura 4.1 muestra de manera gráfica la conexión de los dispositivos de entrada y/o salida en los diferentes nodos que integran la arquitectura física del robot. La distribución de los dispositivos de entrada y/o salida puede variar de acuerdo a la funcionalidad del robot ó del diseñador.

Debido a la importancia que tienen estos dispositivos para el funcionamiento continúo y confiable del robot se pueden encontrar de forma replicada en algún otro nodo (ya sea de tipo múltiple o de tipo único). La replicación de dispositivos de entrada y salida se utiliza comúnmente para tolerar fallos en los sistemas computacionales. La decisión de duplicar los dispositivos no se realiza de manera arbitraria se requiere realizar *análisis de fiabilidad.*

Una manera adicional de tolerar fallos físicamente de estos dispositivos es propuesta en este trabajo y se describe a continuación: Se decidió conectar cada dispositivo (sensor o actuador) en diferentes nodos (mínimo en dos) de tal manera que únicamente en un nodo debe estar en estado ***activo*** y en el otro(s) nodo(s) debe estar en estado ***inactivo ó dormido;*** la razón de realizar esta doble conexión tiene la finalidad de que si llegase a fallar el microcontrolador, la memoria ó la conexión del nodo a la red, los dispositivos de entrada y/o salida conectados al nodo en fallo se pueden activar en otro nodo donde están doblemente conectados, y de esta manera puedan continuar funcionando.

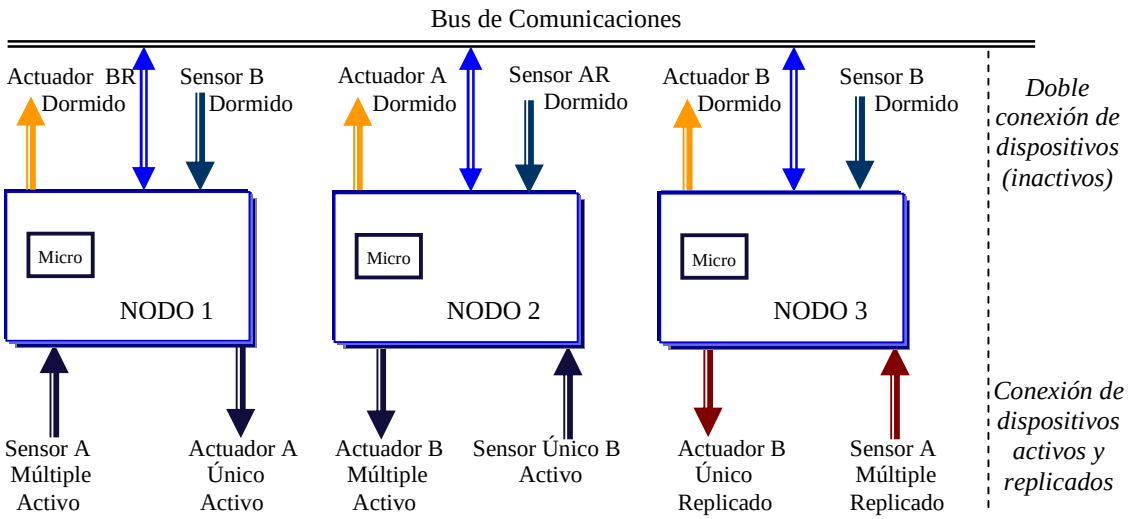


Figura 4. 1 Ejemplo gráfico de distribución de dispositivos de entrada y salida en el sistema de control distribuido de un robot móvil.

- Dispositivos activos y su correspondiente conexión activa.
- Conexión doble de los actuadores (inactivos ó dormidos).
- Conexión doble de los sensores (inactivos ó dormidos).
- Conexión a la red de cada nodo que integra el sistema distribuido.
- Dispositivos replicados en otros nodos (inactivos ó dormidos).

4.2.3 Diferentes tipos de tareas existentes en los nodos del sistema de control

Cualquier sistema computacional esta constituido por una serie de tareas que se implementa para llevar a cabo una determinada funcionalidad, y lograr así sus objetivos. A continuación se enlistan las tareas que realiza generalmente un robot móvil, sin embargo podrán existir otras y estas dependerán de los objetivos que debe cumplir el robot.

- Tareas de control.
- Tareas de funcionamiento del robot (monitoreo, inspección, búsqueda, rescate, reconocimiento, detector de bombas, detector de radioactividad, servir bandejas, soldar, atornillar, etc.).
- Tareas de planeación de acciones.

- Tareas para generar trayectorias.
- Tarea de ejecución de secuencia de movimientos.

En este trabajo se propone tener una tarea copia (***tc***) por cada una de las tareas que constituyen el sistema de control con la finalidad de tolerar los fallos a nivel software; esto dependerá del diseñador y el grado de fiabilidad que se quiera obtener, ya que es posible contar con más de una ***tc***. La estructuración y diseño de las tareas antes mencionadas no forman parte del desarrollo de este trabajo, es responsabilidad del diseñador del robot.

4.3 Tipo de agentes tolerantes a fallos del SMA y su distribución en el sistema de control

Los agentes que integrarán la arquitectura tolerante a fallos se clasifican en los siguientes tipos:

- Agentes tolerantes a fallos para el nivel hardware, designados cómo ***agente nodo***.
- Agentes tolerantes a fallos para el nivel software denominados ***agente tarea*** y ***agente sistema***.

4.3.1 Distribución del agente nodo en sistema de control

El nodo debe de contener los agentes tolerantes a fallos denominados *agentes nodos* de acuerdo a los siguientes criterios:

- Debe de existir un *agente nodo* por cada dispositivo conectado de manera activa (múltiple o único) de entrada/salida.
- Debe de existir un *agente nodo* duplicado en estado inactivo ó dormido por cada dispositivo conectado doblemente al nodo.
- Debe de existir un *agente nodo* duplicado en estado dormido por cada dispositivo replicado.

Los *agentes nodos* que se encuentren inactivos ó dormidos deben estar pendientes de los mensajes que se les envía ya que pueden recibir un mensaje de activación, el responsable enviar este tipo de mensajes es el agente sistema. En la figura 4.2 se muestra un ejemplo gráfico de la distribución de ***los agentes nodo con su estado*** que toleran los fallos de cada dispositivo conectados en los diferentes ***nodos*** que integran el sistema de control distribuido del robot [Ors C.Rafael, Serrano Juan J., Alanis G.A, 2001],[Ors Carot, R., Alexandres G. G, 2002].

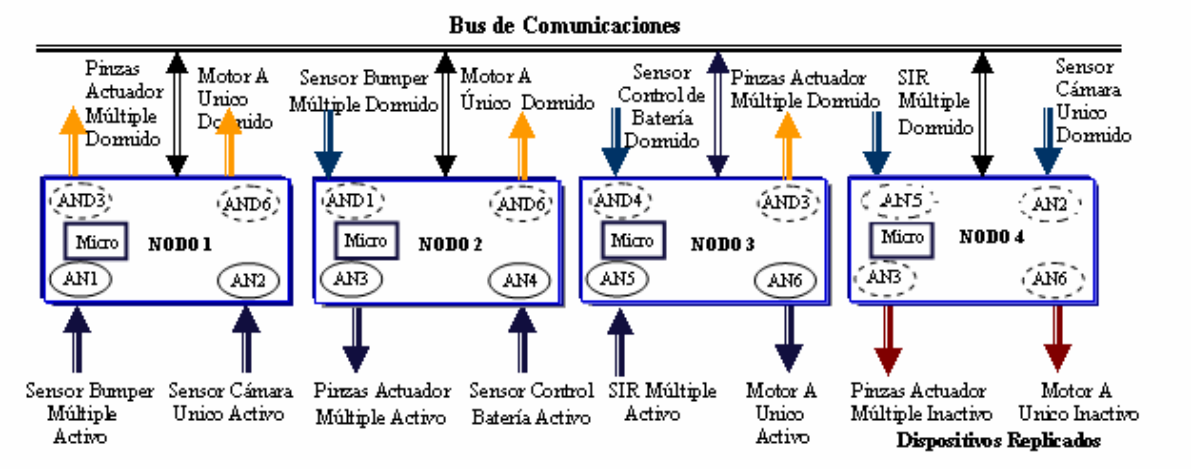


Figura 4. 2 Distribución gráfica de los componentes físicos y agentes nodos sobre la arquitectura física de control distribuido del robot móvil.

En la figura 4.3 se visualiza como encuentran conectados los dispositivos de entrada y salida a cada uno de los nodos de acuerdo con la figura 4.2 tomando en cuenta su estado (activo, dormido).

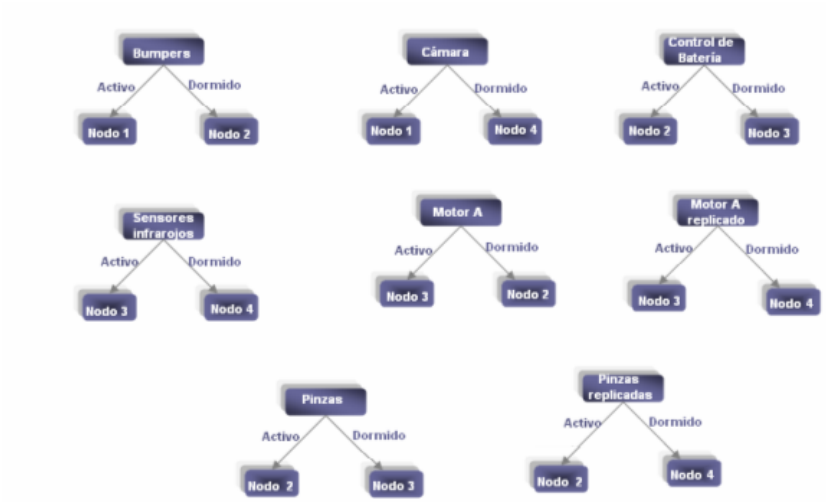


Figura 4. 3 Localización y estado de los dispositivos de entrada y salida en los diferentes nodos que integran la arquitectura física del control distribuido del robot.

4.3.2 Distribución del agente sistema en sistema del control

El *agente sistema* está duplicado en cada uno de los nodos que constituyen el sistema de control, y contienen la misma información todos los nodos su estado es activo solamente en un nodo, y se le denomina Agente Sistema Activo (ASA), en los demás nodo se le denomina Agente Sistema Pasivo (ASP).

4.3.3 Distribución del agente tarea en el sistema de control

El sistema de control se integra por una determinada cantidad de tareas en cada nodo, el número de tareas dependerá de las funciones que debe realizar el robot de acuerdo al objetivo para el cual es diseñado, estas tareas serán repartidas en los diferentes nodos tomando en cuenta los dispositivos que se localizan conectados a los nodos (ver figura 4.4) [Ors C.Rafael, Serrano Juan J., Alanis G.A, 2001],[Ors Carot, R., Alexandres G. G, 2002].

4.3.4 Distribución del agente sistema, agente tarea y agente nodo en un determinado nodo del sistema de control

En la figura 4.4 se muestra los diferentes tipos de agentes tolerantes a fallos que están a cargo de realizar la tolerancia a fallos en los dispositivos y tareas en el nodo siendo para este caso.

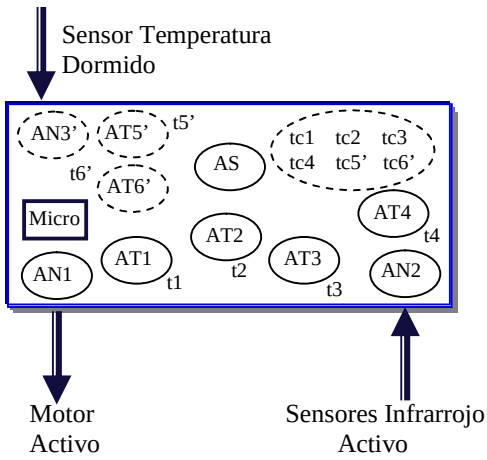


Figura 4.4 Distribución de los agentes que integran la tolerancia a fallos en un nodo que integra el sistema de control del robot.

- El Agente Sistema (AS) tolera fallos del microcontrolador, memoria y controlador de red (estado activo).
- El Agente Nodo 1 (AN1) tolera los fallos del motor único dispositivo de salida (estado activo).
- El Agente Nodo 2 (AN2) tolera los fallos del sensor infrarrojo múltiple dispositivo de entrada (estado activo).
- El Agente Nodo 3 (AN3' estado dormido) tolera los fallos del sensor de temperatura único dispositivo de entrada en doble conexión (estado dormido).
- El Agente Tarea 1 (AT1) tolera los fallos de la tarea 1 (t1 estado activo), y la tarea copia 1(tc1 estado dormido).
- El Agente Tarea 2 (AT2) tolera los fallos de la tarea 2 (t2 estado activo), y la tarea copia 2 (tc2 estado dormido).
- El Agente Tarea 3 (AT2) tolera los fallos de la tarea 3 (t3 estado activo), y la tarea copia 3 (tc3 estado dormido).

- El Agente Tarea 4 (AT4) tolera los fallos de la tarea 4 (t4 estado activo), y la tarea copia 4 (tc4 estado dormido).
- El Agente Tarea 5 (AT5' estado dormido) tolera los fallos de la tarea 5 (t5' estado dormido), y la tarea copia 5 (tc5' estado dormido).
- El Agente Tarea 6 (AT6' estado dormido) tolera los fallos de la tarea 6 (t6' estado dormido), y la tarea copia 6 (tc6' estado dormido).

Como se puede apreciar en la figura 4.4 cada dispositivo y tarea cuenta con un agente tolerante a fallos (agente nodo para los dispositivo activos ó dormidos, agente tarea para las tareas y las tareas copias), dependiendo del estado del dispositivo o tarea su agente debe de tener el mismo estado (los agentes que se localizan en círculos pespunteados en la figura 4.4 su estado es dormido). Cada tarea tiene una tarea copia (o más si así lo decide el diseñador) y su estado debe ser dormido (las tareas copias se localizan en un solo circulo pespunteados en la figura 4.4 su estado es dormido), por cada nodo se debe localizar un agente sistema. En resumen la figura 4.4 muestra los siguientes dispositivos, tareas, y agentes que contiene un nodo del sistema de control: Se localizan conectados al nodo los siguientes dispositivos de control:

- El microcontrolador (activo).
- El controlador de la red (activo).
- Las memorias (activas).

El agente que tolera sus fallos es:

- El Agente Sistema AS.

Se localizan conectados al nodo los siguientes dispositivos de entrada y salida:

- El motor (activo).
- Los sensores infrarrojos (activos).
- Sensor de temperatura (dormido y en doble conexión).

Los agentes nodos que toleran sus fallos:

- AN1 activo para el motor.
- AN2 activos para los sensores infrarrojos.
- AN3' dormido para el sensor de temperatura en doble conexión.

Las tareas activas que se ejecutan en dicho nodo son:

- La tarea uno t1 controla el motor.
- La tarea dos t2 controla los sensores infrarrojos.
- La tarea tres t3 detectar obstáculos.
- La tarea cuatro t4 elabora mapa.

Tareas copias no activas:

- tc1 controla el motor.
- tc2 controla los sensores infrarrojos.
- tc3 detectar obstáculos.
- tc4 elabora mapa.

Cada una de anteriores tareas y sus tareas copias les tolera los fallos un agente tarea activo:

- AT1 tolera los fallos de la t1 y tc1.
- AT2 tolera los fallos de la t2 y tc2.
- AT3 tolera los fallos de la t3 y tc3.
- AT4 tolera los fallos de la t4 y tc4.

También están dentro del nodo dos tareas t5' y t6' en estado dormido y sus tareas copia dormidas tc5' y tc6', estas tareas trabajan con la información que recoge el sensor de temperatura del medio ambiente (su estado es dormido y en doble conexión); estas tareas cuentan con su correspondiente agente tarea dormidos AT5', AT6'. Se tiene una tarea copia dormida por cada tarea activa para garantizar la tolerancia a fallos a nivel software esta tarea copia será activada por el agente tarea en el caso que llegase a tener un fallo la tarea activa.

Agente	Descripción
Agente Sistema (AS)	***Encargado de tolerar fallos del microcontrolador, las memorias y el controlador de la red, y de reconfigurar el sistema de control en caso que falle cualquier dispositivo o tarea.***
Agente Nodo 1 (AN1) activo	***Encargado de la tolerancia a fallos del motor.***
Agente Nodo 2 (AN2) activo	***Encargado de la tolerancia a fallos del sensor múltiple infrarrojo.***
Agente Nodo 3 (AN3) dormido	***A cargo de la tolerancia a fallos del sensor de temperatura en doble conexión, y lo activa si fuera necesario.***
Agente Tarea 1 (AT1)	***A cargo de la tolerancia a fallos de la tarea 1 (t1) y de su copia (tc1), tarea 1 controla al motor y si esta llegase a fallar, el AT1 se encarga de activar a su tarea copia 1 (tc1).***
Agente Tarea 2 (AT2)	***A cargo de la tolerancia a fallos de la tarea 2 (t2) y de su copia tc2, la tarea controla a los sensores infrarrojos, y si llegase a fallar el AT2 se encarga de activar a su tarea copia 2 (tc2).***
Agente Tarea 3 (AT3)	***Encargado de la tolerancia a fallos de la tarea 3 (t3) y de su copia tc3, esta tarea detecta obstáculos, y si llegase a fallar el agente AT3 se encarga de activar su tarea copia tc3.***
Agente Tarea 4 (AT4)	***Encargado de la tolerancia a fallos de la tarea 4 (t4)y de su copia tc4, esta tarea realiza los mapas, y si llegase a fallar el AT4 se encarga de activar su tarea copia tc4.***
Agente Tarea dormido (AT5)	***Se encarga de la tolerancia a fallos de la tarea 5 (t5', dormida) cuando es activada por éste agente y de su copia tc5', en estado activo controla al sensor de temperatura, y si la tarea llegase a fallar en estado activo el AT5 se encarga de activar su tarea copia tc5'.***
Agente tarea dormido (AT6)	***Se encarga de la tolerancia a fallos de la tarea 6 (dormida) cuando la activa (t6'), y si llegase a fallar en estado activo el AT6 se encarga de activar a su tarea copia tc6'.***

Tabla 4. 2 Descripción de los agentes correspondientes según la figura 4.4.

4.3.5 Localización de los dispositivos, tareas y los diferentes tipos de agentes en el sistema de control distribuido del robot móvil

En la figura 4.5 se muestra la arquitectura del sistema de control distribuido del robot con sus dispositivos de entrada/ salida, microcontrolador, tareas, tareas copias, las dobles conexiones de los dispositivos en otros nodos, de tal manera que los agentes tolerantes a fallos de cada uno de los dispositivos y tareas puedan realizar los mecanismos de tolerancia a fallos de una manera efectiva, y el agente sistema de pueda reconfigurar al sistema de control en caso de que ocurra un fallos en el microcontrolador, controlador de la red, memorias, los dispositivos de entrada/salida, y tareas, logrando que a pesar de que ocurra un fallo en software ó hardware se pueda obtener la máxima disponibilidad en el sistema de control en el robot móvil.

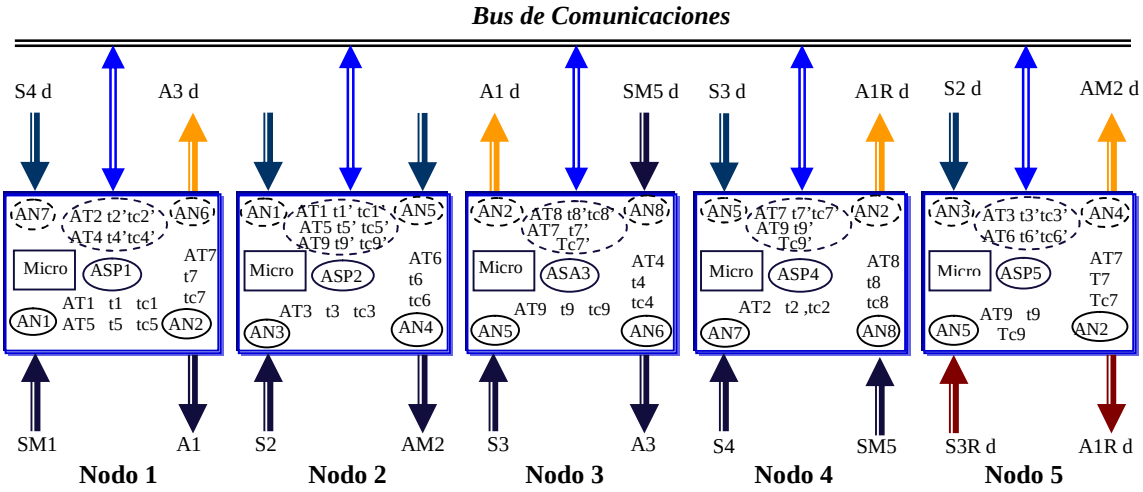


Figura 4.5 Distribución en el sistema de control del robot de los dispositivos replicados y en doble conexión, agentes nodos, las tareas, tareas copias, agentes tareas, el agente sistema activo (ASA) y sus replicas (ASN).

Cada sensor y/o actuador están conectados por lo menos a dos nodos, en un nodo está conectado ***activo*** y en el otro nodo ***dormido*** (d), y en este caso en particular el sensor (*S3*) y el actuador (*A1*) están replicados y ***dormidos***. La Tabla 4.3 muestra el estado activo, dormido y replicado de los dispositivos contenidos en la figura 4.5.

Dispositivo E/S	*Nodo Conexión en estado activo*	*Nodo Doble conexión*	*Nodo Conexión con réplica*	*Nodo Doble conexión de la réplica*
Actuador 1 (A1)	*1*	*3*	*5*	*4*
Actuador Múltiple 2 (AM2)	*2*	*5*	-	-
Actuador 3 (A3)	*3*	*1*	-	-
Sensor Múltiple 1 (SM1)	*1*	*2*	-	-
Sensor 2 (S2)	*2*	*5*	-	-
Sensor 3 (S3)	*3*	*4*	*5*	*2*
Sensor 4 (S4)	*4*	*1*	-	-
Sensor Múltiple 5 (SM5)	*4*	*3*	-	-

Tabla 4.3 Localización de dispositivos conectados activos, dormidos y replicados en los nodos que integran el sistema de control distribuido del robot móvil, mostrado en la figura 4.5.

Los *agentes nodos* localizados en el nodo 1 de la figura 4.5 son:

El agente nodo1 (*AN1)* estado activo tolera los fallos del sensor múltiple1 (*SM1)* estado activo.

El agente nodo2 (*AN2)* estado activo tolera los fallos del actuador1 (*A1)* con estado activo.

El agente nodo7 (*AN7')* estado dormido tolera los fallos del sensor4 (*S4')* en doble conexión y su estado es dormido, se localiza también conectado en el nodo4 con la diferencia que en éste nodo su estado es activo, por consiguiente el *AN7* se localiza en estado activo en el nodo4.

El agente nodo6 (*AN6')* estado dormido tolera los fallos del actuador3 (*A3')* en doble conexión y su estado es dormido, se localiza también conectado en el nodo3 con la diferencia que en éste nodo su estado es activo, por consiguiente el *AN6* se localiza en estado activo en el nodo3.

El nodo1 de la figura 4.5 contiene las siguientes tareas:

Tarea 1 (*t1*) y tarea 5 (*t5*) en estado activo que trabajan con los datos de entrada del sensor múltiple 1 (*SM1*) en estado activo, la tarea 7 (*t7*) en estado activo que trabaja con la señal de salida del actuador 1 (*A1*) en estado activo, cada una de estas tareas cuenta con su respectiva tarea copia en estado dormido siendo estas tc1, tc5, tc7.

La tarea 2 y su copia (*t2' y tc2'*) en estado dormido trabaja con la señal de entrada del sensor cuatro dormido (*S4'*) y en doble conexión, la tarea 4 y su copia *(t4' y tc4'*) en estado dormido trabaja con la señal de salida del actuador 3 (*A3'*) en estado dormido y en doble conexión, en la figura 4.5 se puede visualizar el sensor 4 (*S4 d*) en estado dormido en el nodo1, también se localiza conectado al nodo4 a diferencia del nodo1 en éste nodo su estado activo, el caso del actuador 3 (*A3 d)* en estado dormido en el nodo1, también se localiza conectado al nodo3 en éste nodo su en estado activo.

Los agentes tareas localizadas en el nodo 1 de la figura 4.5 son:

- Agente tarea 1 en estado activo (*AT1*) tolera los fallos de la tarea uno (*t1*) en estado activo, y activa a su tarea copia en estado dormido (*tc1'*)en caso de que la t1 falle .
- Agente tarea 5 en estado activo (*AT5*) tolera los fallos de la tarea cinco (*t5*) en estado activo, y activa a su tarea copia estado dormido (*tc5'*) en caso de que la t5 falle.
- Agente tarea 7 en estado activo (*AT7*) tolera los fallos de la tarea siete (*t7*) en estado activo, y activa a su tarea copia estado dormido (*tc7'*) en caso de que la t7 falle.
- Agente tarea 2 en estado dormido (*AT2'*) cuando está en estado activo en el nodo 1 es porqué está en fallo total el nodo 4 donde se localizaba en estado activo, y el *AT2* la debe de activar y tolera sus fallos, si la tarea llegase a fallar ya en estado activo el *AT2* activa a su copia (*tc2'*).
- Agente tarea 4 en estado dormido (*AT4'*) cuando está en estado activo en el nodo 1 es porque está en fallo total el nodo 3 donde se localizaba en estado activo, entonces el AT4 debe de activar a la tarea 4 que está en estado dormido *(t4'*) y tolerar sus fallos, si la tarea llegase a fallar ya en estado activo el (*AT4*) activa a su copia (*tc4'*).

El *agente sistema* localizado en el nodo1 de la figura 4.5 es: es replicado y dormido ASP1.

En la figura 4.5 puede visualizar que el agente sistema se localiza en estado activo en el nodo3 (ASA), y en el resto de los nodos está replicado y en estado dormido (ASP).

Cada uno de los nodos que constituye el sistema de control del robot esta integrado de la misma manera que el nodo 1, con la excepción del nodo5 que tiene conectado las réplicas del sensor3 (*S3R*) y del actuador1 (*A1R*), en este caso se localizan las dos réplicas en el mismo nodo el diseñador las puede distribuir de acuerdo sus requerimientos.

4.4 Conocimiento y funciones de los agentes tolerante a fallos

4.4.1 Conocimiento que debe poseer el agente sistema y sus copias

El agente sistema debe:

1. Conocer el número de nodos que constituyen el sistema de control.
2. Conocer la distribución de los dispositivos de entrada/salida que se encuentran conectados en estado activo en cada nodo.
3. Conocer en qué nodo se encuentra la doble conexión (estado dormido) de cada uno de los dispositivos de entrada/salida.
4. Conocer en qué nodo se encuentra replicado un determinado dispositivo de entrada/salida y su estado (en caso de existir réplica).
5. Conocer la distribución de los agentes nodos en cada nodo y su estado (activo ó dormido), y conocer de que dispositivo esta encargado cada agente nodo.
6. Conocer la prioridad que tiene asignada cada ASP.
7. Conocer la distribución de las tareas y tareas copia en cada uno de los nodos, y su estado (activo o dormido),
8. Conocer la distribución de los agentes tareas y su estado.
9. Conocer qué tarea tiene asignada cada agente tarea.
10. Conocer si existe dependencia entre las tareas con algún dispositivo.
11. Conocer si una tarea tiene dependencia con otra tarea.
12. Conocer que dispositivo o tarea han fallado y en que momento.
13. Conocer si un dispositivo o tarea que falló se recuperó y en que momento.
14. En que estado está cada dispositivo o tarea (funcionando o en fallo).
15. Conocer en que estado se encuentra cada nodo (funcionando ó en fallo).
16. Conocer si un nodo falló y en que momento.
17. Conocer si el nodo que falló se recuperó y en que momento.

4.4.2 Funciones del agente sistema activo

1. Al recibir el mensaje *RegistrodeASP*(*Nodo)* de un Agente Sistema Pasivo (ASP), paso 1 ejecuta la tarea *AsignarIDalASP()* con la finalidad de otorgarle un número de identificación único (ID), paso 2 ejecutar la tarea *AsignarPrioridadASP()* con el objetivo de asignarle su prioridad (P), paso 3 ejecuta la tarea *AsignarEstadoalASP()* con el propósito de asignarle estado (E, dormido), paso 4 ejecuta la tarea *RegistrarASP(ID,E,P,Nodo)* con el objetivo de anotar en su bitácora de registro: en que nodo se localiza (Nodo), identificador (ID), Prioridad (P) y Estado (E).

2. Al recibir el mensaje *RegistrodeAN(ND,Nodo)* de un Agente Nodo (AN), paso 1 ejecuta la tarea *AsignarIDalAN()* con el objetivo otorgarle su número de identificación único (ID), paso 2 ejecuta la tarea *AsignarEstadoalAN()* con la finalidad de otorgarle su estado (E, activo), paso 3 ejecuta la tarea *RegistrarAN(ID,E,ND,Nodo)* con el propósito de anotar en su bitácora de registro en que nodo se localiza (Nodo), su identificador único (ID), estado (E) y al dispositivo que monitorea (ND).
3. Al recibir el mensaje *RegistrodeAT(NT,nodo)* de un agente tarea (AT), paso 1 ejecuta la tarea *AsignarIDAT()* su objetivo es el de otorgarle un número de identificación único(ID), paso 2 ejecuta la tarea *AsignarEstadoAT()* con la finalidad de otorgarle su estado(E, activo), paso 3 ejecuta la tarea *RegistrarAT(ID,E,NT,Nodo)* con el propósito de anotar en su bitácora de registro en que nodo se localiza (LN) el agente tarea, el identificador único (ID), su estado (E), la tarea a la que monitorea (NT) y su localización (Nodo).
4. Cuando termina el registro de todos los agentes que integran el sistema tolerante a fallos, envía el mensaje *ActualizacionRegistroAgentes*()a todos los ASP, con la finalidad de que actualicen la bitácora de registro.
5. Debe de enviar continuamente el mensaje *EstoyActivoASA()* a todos los ASP, este mensaje tiene la finalidad de comunicarle a cada ASP que el nodo donde se localiza almacenado el ASA continúa activo, ya que si llegase a ocurrir un fallo en dicho nodo, el ASP con mayor prioridad se convierte en ASA.
6. Recibe continuamente el mensaje *NodoActivo(ID)* de cada ASP, éste mensajes tiene el objetivo de comunicarle al ASA que los nodos donde están almacenados los ASP están funcionando.
7. Cuando no recibe el mensaje ***NodoActivo(ID)*** de un determinado ASP debe de ejecutar los siguientes pasos: Paso 1 enviar al ASP mensaje de prueba *EstasActivo(ID)* para confirmar si esta activó el ASP, paso 2 si no responde entonces prueba la funcionalidad del nodo enviando al nodo un mensaje de prueba *PruebaAlNodo(Nodo),* paso 3 si la prueba al nodo da como resultado que el nodo esta en fallo, entonces ejecuta la tarea *DesactivarNodo(Nodo),* cuya función es desactivar al nodo con fallo del sistema de control paso 4 desasactiv a atodos los agentes del nodo en fallo al ejecutar la tarea *DesactivaTodosLosagentes()* y continua, en el paso 9, paso 5 en caso que *PruebaAlNodo(Nodo)* da como resultado que el nodo está funcionando y el mensaje *EstasActivo(ID)* no responde entonces ejecuta la tarea *HacerCopiaASP(ID),* con el propósito de hacer una copia del ASP, paso 6 activa a la copia y desactiva al ASP en fallo al ejecutar la tarea *ActivaCopiaASPDesactivaASPEnFallo(),* paso 7 envía al nodo una copia del ASP en el mensaje *EviarCopiaASP(Nodo*), paso 8 recibe mensaje de confirmación del ASP que llego a su destino, paso 9 ejecuta la tarea *ActualizaBitacoraFalloNodo()* con el objetivo de registrar en la bitácora de fallos al nodo ó el ASP que falló, el momento en que ocurrió dicho fallo y si ya está activo y funcionando la copia del ASP en el nodo correspondiente, paso 10 envía el mensaje *ActualizacionBitacorafalloNodo(ID),*a todos lo ASP para que actualicen su bitácora de fallos. En caso que el falló fuera del nodo y no del ASP, paso 11 reconfigurar los dispositivos del nodo en fallo a los nodos donde esta su doble conexión, y para ello ejecutar la tarea *BuscarlaDobleConexiónDispositivo(ID)* su objetivo es localizar al el número del nodo que contiene la doble conexión del dispositivo, paso 12 envía un mensaje *ActivarDispositivoDobleConexion(ND*) al ASP del nodo que tiene la doble conexión para que ejecute las acciones de activación de agentes nodo, tarea y del dispositivo(se debe enviar un mensaje por cada dispositivo conectado al nodo que falló), paso 13 recibe mensaje de confirmación del ASP *OKRecibiMensajeActivarDispositivoDoble Conexión(ID)* .
8. Recibe el mensaje del ASP *ActualizacionBitacoraDYTDobleConexión(ID),* le comunica que ya se realizo la activación del dispositivo en doble conexión y ejecuta la tarea *ActualizarBitacoraDYtareasDC(ID)* con la finalidad de actualizar su bitácora de fallos.
9. Al recibir el mensaje del ASP *ActualizarASABitacoraFalloTarea(ID),* ejecuta la tarea *ActualizaBitacoraTareaConFallo(ID*).

10. Al recibir mensaje del AT *RecupearacionTarea(ID),* ejecuta la tarea *ActualizarRecuperacióndeTarea(ID),* para registrar en la bitácora de fallos su recuperación y la anota como tarea copia, envía el mensaje *RecuperacionTarea(ID)* a todos ASP.
11. Al recibir mensaje del AT *NoRecuperaciónTarea(ID,* ejecuta la tarea *ActualizarNoRecuperaciónDeTarea(ID),* para registra en la bitácora de fallos que no se recuperó, envía el mensaje a todos los ASP *NoRecuperaciónTarea(ID*).
12. Al recibir el mensaje del AT *ParoSeguroPorFalloTareaCriticaSinCopia(ID),* debido que la tarea en fallo es crítica y no tiene otra copia, paso 1 envía un mensaje al usuario que genere una tarea sin fallo y su copia, borre las que están en fallo, las debe de activar y actualizar al ASA y todos los ASP *GerneraYActivaTareaYCopia(ID),* paso 2 recibe mensaje del usuario que ya esta la copia generada y activada en el lugar adecuado, *TareaYCopiaGeneradaYActivada(ID).*
13. Al recibir mensaje del AT *PonerSisDegradado(TC),* debido a que la tarea no es crítica y no tiene otra copia, paso 1 envía mensaje al usuario para que genere una nueva tarea y su copia, las active y actualice la bitácora de fallos del ASA y todos los ASP, paso 3 recibe mensaje del usuario que la tarea y su copia ya están activadas y actualizadas las bitácoras del ASA y todos los ASP *TareaYCopiaGeneradaYActivada(ID).*
14. Al recibir el mensaje del ASP *ActualizaFalloParcialDM(ID),* ejecuta la tarea *ActualizarBitacoraFPDispMul(ID*).
15. Si recibe el mensaje del ASP *ActualizarBitacoraFalloTotalDispMul(ID*), ejecuta la tarea *ActualizarBitacoraFTDispMul(ID*).
16. Al recibir el mensaje del ASP *ActualizarBitacoraFallosDispNoCriticoSRep(ID),* actualiza la bitácora de fallos ejecutando la tarea *DesactivarDispSinReplicaNOCritico(ID*), envía mensaje externo *ActualizarBitacoraFallosDispNoCriticoSRep(ID).*
17. Si recibe el mensaje de un AN *BuscaReplicaDispMul(ID),* paso 1 busca el nodo donde está el dispositivo duplicado ejecuta la tarea *BuscandoDispReplicadoEnOtroNodo(),* paso 2 envía mensaje al AN del nodo donde está duplicado el dispositivo para que lo active y realice la reconfiguración en el nodo *ActivarDispositivoDuplicado().* Paso 3 recibe mensaje del AN de activación y reconfiguración para que actualice la bitácora de fallos *ActualizaBitacoraFalloDDuplicado(ID)* ,paso 4 ejecuta la tarea *ActualizacionBitacoraFallosDispDuplicado(ID*) para realizar la actualización de la bitácora de fallos, paso 4 envía mensaje a todos los ASP para que actualicen su bitácora de fallos *ActualizaBitacoraFalloDDuplicado(ID).*
18. Si recibe de un AN el mensaje *ParoSeguroPorFalloDispCriticoMulSinReplica(ID),* paso 1e envía mensaje al usuario *ParoSeguroPorFalloDispCriticoMulSinReplica(ID),* el usuario que realizará un paro seguro del sistema de control del robot, paso 2 recibe confirmación del usuario del paro *OkRealizaElParo(),* paso 3 ejecuta la tarea *HacerParoSeguro(),* esta tarea apunta en la bitácora de fallos el dispositivo que al fallar provocó el paro del sistema , el momento del fallo, guarda la información que tenga en la memoria el ASA y en su cola de mensajes y realiza un paro seguro al sistema de control.
19. Si recibe mensaje del AN *ParoSeguroPorFalloDispCriticoUnicoSinReplica(),*paso 1 envía mensaje al usuario de paro *ParoSeguroPorFalloDispCriticoUnicoSinReplica(),* paso 2 recibe confirmación de paro por el usuario *OkRealizaParo(),* paso 3 ejecuta la tarea *HacerParoSeguro()*esta tarea apunta en la bitácora de fallos el dispositivo que al fallar provocó el paro del sistema , el momento del fallo, guarda la información que tenga en la memoria el ASA y en su cola de mensajes y realiza un paro seguro al sistema de control.
20. Si recibe de un AN el mensaje *BuscarReplicaDispUnico()* paso1 busca el nodo donde está el dispositivo duplicado ejecuta la tarea *BuscandoDispReplicadoEnOtroNodo(),* paso 2 envía mensaje al AN del nodo donde está duplicado el dispositivo para que lo active y realice la reconfiguración en el nodo *ActivarDispositivoDuplicado().* Paso 3 recibe mensaje del AN de activación y reconfiguración para que actualice la bitácora de fallos *ActualizaBitacoraFalloDDuplicado(ID),* paso 4 ejecuta la tarea

ActualizacionBitacoraFallosDispDuplicado(ID) para realizar la actualización de la bitácora de fallos, paso 4 envía mensaje a todos los ASP para que actualicen su bitácora de fallos *ActualizaBitacoraFalloDDuplicado(ID).*

21. Envía mensaje de activación a un AN dormido con el propósito que active a un dispositivo replicado en otro lado *ActivarANDeDispositivoDuplicado(),* paso 1 recibe mensaje del AN de actualización de bitácora de fallos ya que está activo y funcionando el dispositivo replicado *EstaActivadoDispositivoPorReplica(),* paso 2 ejecuta tarea de actualización de bitácora de fallos *ActualizaciónBitacoraDeFallosDispDuplicado(),* paso 3 envía mensaje a todos los ASP para que actualicen su bitácora de fallos por activación del dispositivo replicado *ActualizarBitacora FallosDispositivoDuplicado(ID).*
22. Envía mensaje de activación a un AN dormido con el propósito que active a un dispositivo en su doble conexión en otro lado *ActivarANDeDispositivoDobleConexión(ID),*paso 1 recibe mensaje del AN de confirmación de que está activo y funcionando el dispositivo en su doble conexión Esta*ActivadoDispositivoporDobleConexion(),* paso 2 ejecuta tarea de actualización de bitácora de fallos *ActualizaciónBitacoraDeFallosDobleConexion(),* paso 3 envía mensaje a todos los ASP para que actualicen su bitácora de fallos por activación del dispositivo en su doble conexión *ActualizarBitacora FallosDispositivoDobleconexion(ID).*
23. Al recibir el mensaje del AN *RecuperciónDisp(ID*), paso 1 ejecuta la tarea *ActualizaRecuperaciónDispositivo(ID*), para anotar en la bitácora que el dispositivo se recuperó, envía el mensaje a todos los ASP para que actualicen su bitácora de fallos *ActualizaRecuperaciónDispositivo(ID*).
24. Si recibe el mensaje del AN *NoRecuperaciónDisp(ID),* ejecuta la tarea *NoFueRecuperadoDisp(ID)* cuya finalidad es anotar en la bitácora que el dispositivo no se recuperó y envía el mensaje de *NoRecuperaciónDisp(ID)* a todos los ASP .

4.4.3 Funciones del agente sistema pasivo

1. Debe de registrarse ante el ASA enviando el mensaje *RegistrodeASP(LD,ID)* al momento de activarse el sistema de control del robot.
2. Al recibir el mensaje del ASA *ActualizacionRegistroAgentes()* debe ejecutar la tarea *ActualizarAgentes(ID),* esta tarea hace una copia de la bitácora de registro del ASA.
3. Debe enviar continuamente el mensaje *NodoActivo(ID)* al ASA indicándole que esta activo.
4. Debe recibir continuamente del ASA mensaje *EstoyActivoASA().*
5. Si no recibe el mensaje del ASA *EstoyActivoASA* (se debe a que el nodo donde estaba almacenado el ASA falló ò fallo el ASA), paso 1 envía mensaje al ASA de confirmación entonces de actividad *EstasActivo(),* paso 2 si el ASA no responde, se manda un mensaje al Nodo para comprobar si está activo *ComprueveActividadDeNodo(),* paso 3 si el nodo responde que esta activo busca prioridad ejecutando la tarea *BuscarPrioridad()* paso 4 localizada su prioridad ejecuta la tarea *ComparaPrioridad()* para saber si él tiene la mayor prioridad si es así ejecuta la tarea *HcerCopiaDeSiMismo(),* realiza un acopia de si mismo para enviarla al nodo donde el ASA está en fallo, paso 5 envía la copia al nodo donde el ASA está en fallo *EnviarCopiaASA(),* paso 6 activa la copia del ASA al ejecutar la tarea *ActivarCopiaDel ASA(),* paso 7 recibe mensaje de comprobación que llegó bien la copia del ASA al nodo *indicado CopiaDelASAListo(),* paso 8 actualiza la bitácora de fallos debido al fallo del ASA *ActualizarBitacoraFalloPorFalloASA(),* paso 9 Envía mensaje a todos los aSP y al ASA de actualización de bitácora *ActualizarBitacoraFalloPorFalloASA().*
6. Si el nodo no responde que está funcionando y el ASA no responde se debe de realizar lo siguiente: paso 3 si el nodo responde que esta activo busca prioridad ejecutando la tarea *BuscarPrioridad(),* paso 4 localizada su prioridad ejecuta la tarea *ComparaPrioridad()* para

saber si él tiene la mayor prioridad si es así ejecuta la tarea *ActualizarASA* para convertirse en el ASA, paso 5 ejecuta la tarea *DesactivarNodo(ID)* que tiene la finalidad de desactivar al nodo en fallo del sistema de control, paso 6 envía el mensaje *SoyNuevoASA* a todos los ASP, paso 7 ejecuta la tarea *ActualizaBitacoraFalloNodo(ID)*, registra en la bitácora de fallos la desactivación del nodo en fallo y el momento en que ocurrió el fallo, paso 8 inicia tareas de reconfiguración de todos los dispositivos que se encontraban activos en el nodo con fallo, al nodo donde tienen su doble conexión (ver funciones del ASA para activación de dispositivos a su doble conexión), paso 9 Recalcula prioridad para cada ASP *RecalcularPrioridades()*, paso 10 envía mensaje de nueva prioridad a cada ASP *nuevaPrioridad(ID)*, paso 11 actualiza bitácora de registro con nueva prioridad a cada ASP *ActualizarBitacoraRegistroNuevaPrioridad()*, paso 12 envía mensaje a todos los ASP *ActualizarBitacoraRegistroNuevaPrioridad()*.

7. Si llegase a recibir mensaje *SoyNuevoASA* ejecuta la tarea *ReconfiguraraNuevoASA()* con el propósito de se direccionarse con el nuevo ASA.
8. Al recibir el mensaje del AT *PonerEnFalloTarea(ID)* debido a que una determinada tarea fallo, paso 1 desactivar a l atarea en fallo *DesactivarTarea En Fallo/(ID)*, paso 2 enviar el mensaje de desactivación de tarea en fallo a todos los ASP y ASA ActualizacionTareaEnFallo(ID).
9. Envía mensaje de activación al AT para que éste a su vez active la debido a que se requiere activar un dispositivo replicado en el nodo donde el AT está dormido *ActivarAgenteTareaPorDispDuplicado(ID)*.
10. Envía mensaje de activación al AT para que éste a su vez active la debido a que se requiere activar un dispositivo en doble conexión en el nodo donde el AT está dormido *ActivarAgenteTareaPorDispDobleCon(ID)*.
11. Recibe mensaje del su AT que a se activó y él a su vez activó: a su tarea y la tarea copia, por fallo en nodo y se requiere activar al dispositivo en su doble conexión *EstaActivadaTareaPorDDConexión(ID)*.
12. Recibe mensaje del su AT que se activo, y él a su vez activó: a su tarea y la tarea copia por fallo en nodo y se requiere activar al dispositivo duplicado *EstaActivadaTareaPorreplica(ID)*.

4.4.4 Conocimiento que debe tener el agente nodo

El agente nodo debe:

1. Conocer en qué nodo se encuentra activo el ASA.
2. Conocer qué ASP esta a cargo de él.
3. Conocer a qué dispositivo de entrada y/o salida le tolera los fallos y su tipo. En caso de tolerarle los fallos a un dispositivo múltiple debe saber el tope mínimo permisible para que este dispositivo funcione de manera confiable.
4. Conocer a los agentes tareas relacionado con él.
5. Conocer el número de nodo donde se encuentra dormido.
6. Conocer en qué otro nodo se encuentra en estado dormido (si es por doble conexión o por réplica).

4.4.5 Funciones del agente nodo

1. Debe de registrarse ante el Agente Sistema (ASA) enviando el mensaje *RegistrodeAN(ID)* al momento de activarse el sistema de control del robot.
2. El Agente Nodo (AN) debe de monitorear continuamente al dispositivo a su cargo con la finalidad de detectar un fallo, para ello ejecuta la tarea *MonitorearDispositivo(ID*), puede detectar el fallo en un dispositivo múltiple o en un dispositivo único.
3. Si detecta el fallo en un dispositivo múltiple: ***Caso 1***: paso 1 el AN ejecutará la tarea *VerificarDispMul(ID*) con el objetivo de verificar la cantidad de dispositivos que le quedan aún activos y compararlos con el máximo permisible. Si el dispositivo múltiple a pesar de que un elemento le haya fallado está dentro del rango para seguir operando correctamente, debe de ejecutar la tarea *DescontarUnSoloDispPorFallo(ID)* esta tarea descuenta el elemento en fallo del conjunto que integra al dispositivo múltiple (entre varios del mismo tipo) el resultado lo pasa a la tarea *VerificarDispMul(ID*) al ejecutar la tarea *PasoDeParametro(ID,N),* paso 2 debe de desactivar únicamente al elemento que falló poniéndolo en estado de fallo e inactivo por lo que requiere ejecutar la tarea *DesactivarSoloUnElemntoDelDispMulEnFallo(ID),* paso 3 envía un mensaje interno a su ASP *ActualizaBitacoraFalloParcialDM(ID*) con el objetivo que el ASP actualice la bitácora de fallo. ***Caso 2***: paso 1 el AN ejecutará la tarea *VerificarDispMul(ID*) con el objetivo de verificar la cantidad de dispositivos que le quedan aún activos y compararlos con el máximo permisible, Si el dispositivo múltiple al fallarle un elemento sobrepasa el rango para seguir operando, debe de ejecutar la tarea *DescontarTodoDispMulPorFallo(ID*) esta tarea descuenta únicamente al elemento en fallo del conjunto que integra al dispositivo múltiple (entre varios del mismo tipo) el resultado lo pasa a la tarea *VerificarDispMul(ID*), paso 2 debe de desactivar a todo el dispositivo múltiple y ponerlo en estado de fallo e inactivo por lo que requiere ejecutar la tarea *DesactivarTodoDispMulEnFallo(ID),* paso 3 debe de enviar el mensaje a su ASP *ActualizaBitacoraFalloTotalDM(ID),* con el objetivo que el ASP actualice la bitácora de fallos.
4. Cuando el AN desactiva el dispositivo múltiple por completo, busca si éste es crítico y no tiene réplica en otro nodo, por lo que debe de ejecutar la tarea *BuscarReplicaCriticoDispMul(ID),*si el resultado de la búsqueda da positivo envía al ASA el mensaje *ParoSeguro PorFalloDispCriticoMulSinReplica(ID).*
5. Si el resultado es negativo después de ejecutar la tarea *BuscarReplicaCriticoDispMul(ID),* busca si el dispositivo tiene réplica en otro nodo, y ejecuta la tarea *BuscarReplicaDispMul(ID),* si el resultado de la búsqueda es positivo envía un mensaje al ASA *BuscaLaReplicaDispMul(ID)* para que el ASA reconfigure al sistema de control poniendo en funcionamiento a la réplica en otro nodo einicia la tarea de recuperación.
6. Si el resultado es negativo después de ejecutar la tarea *BuscarReplicaDispMul(ID),* queda en estado de fallo y desactivo, inicia la tarea para recuperarlo.
7. En el caso que el AN detecte el fallo en un **dispositivo tipo único** paso 1 desactivar el dispositivo y ponerlo en estado de fallo e inactivo por lo que requiere ejecutar la tarea *DesactivarDispUnicoEnFallo(ID),* paso 2 enviar el mensaje a su ASP *ActualizaBitacoraFalloDispUnico(ID),* con el objetivo que el ASP actualice la bitácora de fallos. Paso 3 busca si el dispositivo es crítico y no tiene réplica en otro nodo, por lo que debe de ejecutar la tarea *BuscarReplicaCriticoDispUnico(ID),*si el resultado de la búsqueda es positivo envía al ASA el mensaje *ParoSeguroPorFalloDispCriticoUnicoSinReplica(ID).*
8. Si el resultado es negativo después de ejecutar la tarea *BuscarReplicaCriticoDispUnico(ID),* busca si el dispositivo tiene réplica en otro nodo, y ejecuta la tarea *BuscarReplicaDispUnico(ID),* si el resultado de la búsqueda es positivo envía un mensaje al ASA *BuscaLaReplicaDispUnico(ID)* para que el ASA reconfigure al sistema de control poniendo en funcionamiento a la réplica en otro nodo e inicia la recuperación del dispositivo..

9. Si el resultado es negativo después de ejecutar la tarea *BuscarReplicaDispUnico(ID)*, queda en estado de fallo y desactivo, inicia la tarea para recuperarlo.
10. ***Caso 1*** el agente nodo está ***"dormido"*** y recibe el mensaje *ActivarANDeDispositivoDuplicado(ID*) que proviene del ASA, el mensaje tiene el objetivo de activar al agente nodo para que éste active al dispositivo replicado y tolere sus fallos; paso 1 activación del agente nodo dormido al ejecutar la tarea *ActivarAgenteNodoDormido(ID)* lo pone activo y en funcionamiento, paso 2 el AN activa el dispositivo replicado ejecutando la tarea *ActivarDispositivoReplicado(ID)* poniéndolo en estado activo y en funcionamiento, paso 3 envía mensaje de activación a todos los agentes tareas de las tareas que requiere la señal del dispositivo para operar *ActivarAgenteTareaPorDispDuplicado(ID*), paso 4 enviar el mensaje *EstaActivadoDispositivoPorReplica(ID*) a su ASP para que actualice la bitácora de fallos.
11. ***Caso 2*** el agente nodo está ***"dormido"*** y recibe el mensaje *ActivarANDeDispositivoDobleConexión(ID)* que proviene del ASA, el mensaje tiene el objetivo de activar al agente nodo para que éste active al dispositivo en su doble conexión y tolere sus fallos; paso 1 activación del agente nodo dormido al ejecutar la tarea *ActivarAgenteNodoDormido(ID)* lo pone activo y funcionando, paso 2 el AN activa el dispositivo en su doble conexión ejecutando la tarea *ActivarDispositivoEnDobleConex(ID)* poniéndolo en estado activo y en funcionamiento, paso 3 envía mensaje de activación a todos los agentes tareas de las tareas que requiere la señal del dispositivo para operar *ActivarAgenteTareaPorDispDobleConexion(ID*), paso 4 enviar el mensaje *EstaActivadoDispositivoPorDobleconexion(ID)* a su ASP para que actualice la bitácora de fallos, inicia la ejecución de las tareas tolerantes a fallos del dispositivo en ese nodo.
12. Debe de realiza el proceso de recuperación en cualquier tipo de fallo, por lo implica realizar tareas de *RecuperaciónDelDispositivoEnFallo(ID)*, si el dispositivo se recupera envía mensaje *RecuperciónDisp(ID)* al ASA, si no se logra su recuperación envía el mensaje de *NoRecuperaciónDisp(ID)* al ASA para que lo ponga en fallo permanente.

4.4.6 Conocimiento que debe tener el agente tarea

El agente tarea (AT) debe:

1. Conocer en qué nodo se encuentra el ASA
2. Conocer el ASP a cargo del nodo donde el agente tarea se localiza.
3. Conocer sobre qué tarea opera, así como la información referente a su o sus tareas copias (número de tareas copias que tenga).
4. Conocer a las tareas (si es el caso) que están relacionadas con la tarea que monitorea.
5. Conocer a lo(s) agente(s) nodo(s) a cargo del dispositivo (si es el caso) con el que tiene relación la tarea que monitorea.

4.4.7 Funciones del agente tarea

1. Debe de registrarse con el Agente Sistema (ASA) y enviarle el mensaje externo *RregistrodeAT(ID)* en el momento de activarse el sistema de control del robot.
2. El agente tarea (AT) debe de monitorear continuamente la tarea a su cargo con la finalidad de detectar algún fallo por lo que debe ejecuta la tarea *MonitorearTarea(ID*).
3. Si detecta un fallo debe de realizar lo siguiente: envía mensaje interno *PonerEnFalloTarea(ID)* a su ASP.

4. Realiza el proceso de aislamiento que implica desactivar la tarea con fallo, por lo que requiere ejecutar la tarea *DesactivarTarea(ID),* debe de activar a su tarea copia *ActivaTareaCopia(ID*), algunas tareas requieren para su funcionamiento información de los dispositivos de entrada/salida por lo que debe de buscar al AN si la tarea que falló tiene relación con algún dispositivo para ello debe de ejecutar la tarea *BuscarRelaAgenteNodo(ID),* si existe relación con algún dispositivos de entrada/salida envía un mensaje interno al agente(s) nodo(s) (AN) con la finalidad de que éste envié las señales de los dispositivos a la tarea copia *DireccionarDispositivoATareaCopia(ID),* debe de recibir mensaje interno del AN de confirmación del direccionamiento *RealizadoDireccionamientoALaTareaCopia(ID).*
5. Algunas tareas requieren para su funcionamiento información de otras tareas, por lo que debe de buscar a l AT si la tarea que falló tiene relación con alguna otra tarea; para ello debe de ejecutar la tarea *BuscarRelAgenteTarea(ID),* si existe relación con alguna tarea envía un mensaje interno al agente(s) tareas(s) (AT) con el objetivo de que éste envié la información a la tarea copia *direccionarTareaATareaCopia(ID),* debe de recibir mensaje interno de confirmación AT que se realizó el direccionamiento *RealizadoDireccionamientoALaTareaCopia(ID).*
6. envía el mensaje interno a su ASP de *ActualizarBitacoraFallosPorTarea(ID),* con la finalidad de que el ASP actualice su bitácora de fallos; anotando en estado de fallo a la tarea con fallo, en estado activo a la tarea copia .
7. Realiza el proceso de recuperación de la tarea, lo que implica ejecutar la tarea *RecuperaciónTareaFallo(ID*) , si la tarea se recupera, la pone como tarea copia y ejecutando la tarea *EsTareaCopia(ID),* y envía mensaje *RecupearacionTarea(ID)* al ASA, si no logra su recuperación entonces le envía al ASA el mensaje *NoRecuperaciónTarea(ID).*
8. En caso de ser un agente tarea dormido recibe cualquiera de los dos mensajes internos *ActivarAgenteTareaDispDobleCon(ID)* ó *ActivarAgenteTareaPorDispDuplicado(ID*), de un AN (el mensaje tiene la finalidad de activa la tarea y tolerar sus fallos, ya sea porque se requiere conectar un dispositivo en su doble conexión, o se necesita activar un dispositivo réplica), paso 1 ejecutar la tarea *ActivarAgenteTarea(ID*), con la finalidad de activar al AT, paso 2 el AT activa a su tarea e inicializa las funciones de tolerancia a fallos, por lo que requiere ejecutar la tarea dependiendo del caso *ActivarTareaPorDD(ID)* ó *ActivaTareaPorReplica(ID),* paso 3 enviar a su ASP un mensaje interno *EstaActivadaTareaPorDD(ID)* ó *EstaActivadaTareaPorReplica(ID)* para que se actualice la bitácora de fallos.

4.5 Comunicación entre los agentes tolerante a fallos

Las Tablas (4.4, 4.5 y 4.6) muestran las situaciones en las que cada tipo de agente se comunica con los otros tipos de agentes.

DEL AGENTE NODO	
Comunicación	*Situación*
Agente Nodo	**No existe**
Agente Sistema Activo	**1. Al momento de iniciar el sistema de control y activarse por primera vez, le debe informar al ASA que lo registre.** **2. Cuando debe de activar la réplica de un dispositivo en otro nodo** **3. Cuando no se pudo recuperar el dispositivo** **4. Al recuperarse un dispositivo**
Agente Sistema Pasivo	**1. Cuando ocurre un fallo en el dispositivo que monitorea.** **5. Cuando termina de realizar el aislamiento del fallo en el dispositivo que tiene a su cargo** **6. Cuando activa o duerme al dispositivo que tiene a su cargo**
Agente Tarea	**1. Envía mensaje de desactivación a la tarea que tiene a su cargo.** **2. Enviar mensaje de activación a la tarea que tiene a su cargo**

Tabla 4. 4 Comunicación del agente nodo con otros agentes.

AGENTE SISTEMA	
Comunicación	*Situación*
Agente Nodo	**1. Cuando le indica que active el dispositivo a su cargo que se encuentra dormido ya sea por que está en doble conexión o replicado.** **2. Cuando le indique que duerma el dispositivo a su cargo porque ya se recupero se ha recuperado un dispositivo en el nodo X y en el nodo Y se encuentra en función dicho dispositivo, el AS del nodo Y le indica al AN en el nodo Y que duerma al dispositivo, el AS del nodo X se comunica con el AN en dicho nodo indicándole que active la señal de dicho dispositivo.**
Agente Tarea	**1. Al momento de fallar la tarea copia en el Nodo X y de no poderse recuperar, el AS (del nodo Y) se comunica con el AT del Nodo Y en el que se encuentra replicada dicha tarea para que active su correspondiente tarea.** **2. Al momento de recuperarse la tarea en el Nodo X, el AS (del nodo Y) se comunica con el AT del Nodo Y en el que se encuentra replicada dicha tarea para que desactive su correspondiente tarea.**
Agente Sistema	**1. Se comunica el ASA con el ASP para indicarle que se encuentra vivo, de la misma manera el ASP con el ASA para indicarle que se encuentra activo.** **2. El ASP se comunica con el ASA para indicarle de los fallos que han ocurrido en el nodo en que se encuentra con la finalidad de que se mantenga la bitácora actualizada.** **3. Cuando existe algún cambio que realizar debido algún fallo en algún nodo, el ASA se comunica con los ASP indicándoles de dicho cambio para que actualicen su base de conocimiento.** **4. Los ASP le indican al ASA cuando se ha dormido o activado algún dispositivo o tarea en un nodo, de la misma manera, el ASA le indica al ASP cuando debe dormir o activar un dispositivo o tarea.**

Tabla 4 5 Comunicación del agente sistema con otros agentes.

AGENTE TAREA		
Comunicación		*Situación*
Agente Nodo	**1.**	**Al momento de fallar una tarea y activar la tc, el AT le informa al AN encargado del control del dispositivo con el que trabaja dicha tarea que reconfigure la señal del dispositivo con la tc.**
Agente Sistema Activa	**1.** **2.**	**Cuando ocurre un fallo, ya sea con la tarea o tc.** **Cuando el AT ha recuperado la tarea o tc.**
Agente Sistema Pasivo	**1.**	**Cuando el AT activa o duerme a una tarea o tc.**
Agente Tarea	**1.** **2.** **3.**	**En caso de ocurrir un fallo en una tarea, el AT encargado de dicha tarea le deberá comunicar a los AT de las tareas que dependen de dicha tarea que reconfiguren a sus respectivas tareas con la tc.** **Cuando el AT recupera la tarea, le comunicará a los AT dependientes de dicha tarea que se reconfiguren ahora con ella.** **Cuando falla una tc y no se recupera, el AT le indicará a los AT de las tareas que dependen de la tc que deberán desactivar sus respectivas tareas.**

Tabla 4. 6 Comunicación del agente tarea con los otros agente

La tabla (4.7) muestra de manera abstracta los tipos de agentes empleados para tolerar fallos para un sistema de control para un robot móvil.

Tolerancia a Fallos a Nivel	*Tipo de Agente*
Software	▪ **Un agente sistema activo (ASA) en un solo nodo.** ▪ **Un agente sistema pasivo (ASP) por cada nodo restante que integra el arquitectura física de control distribuido para el robot móvil.** ▪ **Un AT para cada una de las tareas (activa o dormida) que integran el sistema de control y su respectiva tarea copia.**
Hardware	▪ **Un agente nodo (AN) por cada tipo de dispositivo de entrada y salida existente en un nodo, ya sea en estado activo o dormido.**

Tabla 4. 7 Tipo de agente y su nivel de tolerancia a fallos.

En resumen la tolerancia a fallos en el nodo se lleva a nivel hardware y a nivel software de la siguiente manera (tabla 4.8).

Tolerancia a Fallos a Nivel	*Replica*
Software	▪ **Replicando las tareas existentes en el sistema de control del robot, esta replica pasará a ser una tarea copia (tc).** ▪ **Replicando al agente sistema en todos los nodos del sistema de control del robot.** ▪ **Duplicando a los agentes nodos de cada dispositivo en los nodos que contienen la doble conexión y replica de cada dispositivo.** ▪ **Duplicando los agentes tareas de cada tarea relacionada con las dobles conexiones y replicas de los dispositivos.**
Hardware	▪ **Replicando dispositivos (sensores, actuadotes).** ▪ **Conectando un mismo dispositivos a un nodo diferente.**

Tabla 4. 8 Resumen de niveles de tolerancia de fallos en el AFCDRM.

4.6 Descripción formal del sistema tolerante a fallos

Sea *SD*, un sistema distribuido compuesto por un conjunto de *nodos* $N = \{N_i\}$, donde cada N*i* puede estar formado por varios *dispositivos* $[D_{i,z}]$. Por otra parte, sobre SD se ejecuta un conjunto de *tareas*, $T = \{T_j\}$.

Definición 1: Sea $N = \{N_i\}$, donde *i* es el número de nodos del sistema distribuido.

Definición 2: Sea $T = \{T_j\}$, donde *j* es el numero de tareas que se ejecutan en el sistema.

Definición 3: Sea $[D_{i,z}]$, donde *z* es el número de dispositivos que tendrá N_i . A partir de estas definiciones, se puede realizar las siguientes definiciones:
Definición 4: Sea un sistema distribuido SD formado por la dupla: SD = $\{N, T\}$

A dicho SD se le pretende dotar de ciertas características de tolerancia a fallos. Para ello se propone la utilización del paradigma de la IAD, con lo que se puede hablar de un nuevo enfoque los SFT en SD con la implementación de Agentes [Ors C.Rafael, Serrano Juan J., Alanis G.A, 2001],[Ors Carot, R., Alexandres G. G, 2002]

$$\mathbf{AITF = \{AN_i, AT_j, AS\}}$$

Se definirán ahora los Agentes Tolerantes a fallos, que trabajaran en *SD.*

El Agente Nodo $(AN_i) \in N_i$, cuya misión es la relacionada con la tolerancia a fallos a nivel de nodo (que funciona y que no dentro del nodo).

El Agente Tarea $(AT_j) \in T_j$, cuya misión es la relacionada con la tolerancia a fallos a nivel de tarea (como recuperar las tareas de los posibles errores que puedan sufrir).

El Agente Sistema (AS) $\in$ *S*D, cuya misión es la relacionada con la tolerancia a fallos a nivel de sistema (qué tareas deben ejecutarse en el sistema y sobre qué nodos). Con ello un SD tolerante a fallos se define como:

Definición 5: Se define un Sistema Distribuido Tolerante a Fallos SDTF como la dupla:

$$\mathbf{SDSTF = \{SD, AITF\}}$$

A continuación se pasas a describir en mayor detalle cada uno de los agentes que componen el AITF.

4.6.1 Agente Nodo (AN_i)

El agente nodo es el encargado de realizar todas las acciones encaminadas a aportar la Tolerancia a Fallos a nivel de nodo. Internamente cada Agente Nodo contempla una serie de variables, definidas de forma estándar, que gobiernan su funcionamiento [Ors C.Rafael, Serrano Juan J., Alanis G.A, 2001],[Ors Carot, R., Alexandres G. G, 2002].

Definición 6: Sea (AN$_i$).Fase. Es una variable que puede obtener cinco hechos: ***Detección, Localización, Aislamiento, Reconfiguración y Recuperación.***

Donde:
(AN$_i$).Fase.*Detección*. En esta fase de detección de fallos se supone que el nodo funciona correctamente.

(AN$_i$).Fase *Localización.* A esta fase, se entra tras la detección de un error y se pretende localizar el dispositivo causante del mismo.

(AN$_i$).Fase *Aislamiento.* En esta fase se intenta aislar el dispositivo en fallo.

(AN$_i$).Fase *Reconfiguración*. En esta fase se efectúa la reconfiguración de los dispositivos del nodo.

(AN$_i$).Fase *Recuperación*. En esta fase se lleva a cabo la recuperación del error.

Cada nodo, bajo sus circunstancias constructivas, puede contemplar diferentes mecanismos de detección de errores. Estos mecanismos pueden ser hardware, software (test) o una combinación de ambos. Todos estos mecanismos de detección de errores, generan una información acerca del estado del sistema que es recogida directamente por el AN$_i$ [Ors C.Rafael, Serrano Juan J., Alanis G.A, 2001],[Ors Carot, R., Alexandres G. G, 2002].

Definición 7: Sea (AN$_i$).Entrada-Error($_{i,j}$). Es una variable que se activa por los diferentes mecanismos de detección de errores implementados es AN$_i$. Tiene dos hechos asociados: ***LibredeError*** **y** ***Error***. En la inicialización se supone que todas las entradas están ***LibredeError,*** de forma que cuando un mecanismo de detección de error (ya sea hardware o software, para los dispositivos que conforman el N$_i$) de los j mecanismos de detección de errores presentes en el AN$_i$ detectan un error, esta variable pasa a la situación (AN$_i$).Entrada-Error($_{i,j}$).Error ($_i$ número de nodo, $_j$ es tipo de mecanismo de detección de error implementado en el AN$_i$) .

Definición 8: Sea (AN$_i$).Estado. Es una variable indica el estado del N_i y tiene cuatro hechos: ***Sospechoso, Correcto, Degradado, Baja***.

Donde:

(AN$_i$).Estado.*Sospechoso*. Este estado indica que *Ni*, posiblemente tiene un fallo.

(AN$_i$).Estado.*Correcto.* Este estado indica un funcionamiento del N_i libre de fallos.
(AN$_i$).Estado.D*egradad.* Este estado indica que en el N_i existe algún dispositivo D$_{i,z}$ que está estropeado(i número del nodo, z número de dispositivo en el N_i).

(AN$_i$).Estado.*Baja*. Este estado indica que él N_i no funciona.

Definición 9: Sea (AN$_i$).Test[D$_{i,z}$]. Es una variable es activada cuando se realiza un test al dispositivo $_z$ del N$_i$ con las acciones implementadas en el AN$_i$ de realización de test para dicho dispositivo. Y esta variable tiene 2 hechos asociados ***Pasa y No-Pasa.***

Donde:

(AN$_i$).Test[D$_{i,z}$].Pasa. Indica que el dispositivo z del nodo i pasó el test que le aplicó el AN.

(AN_i).Test[$D_{i,z}$].No-Pasa. Indica que el dispositivo z del nodo i no pasó el test que le aplicó el AN.

Definición 10: Sea (AN_i).Dispositivo[$D_{i,z}$]. Es una variable que indica el estado del dispositivo $_z$ del nodo $_i$. Esta variable tiene tres hechos asociados. ***Correcto, Incorrecto o Baja***.

Donde:
(AN_i).Dispositivo[$D_{i,z}$].Correcto. Indica que el dispositivo $_z$ del nodo $_i$ tiene un funcionamiento correcto.

(AN_i).Dispositivo[$D_{i,z}$].Incorrecto. Indica que el dispositivo $_z$ del nodo $_i$ no responde correctamente.

(AN_i).Dispositivo[$D_{i,z}$].Baja. Indica que el dispositivo $_z$ del nodo $_i$ tiene un fallo permanente y el AN_i lo dio de baja.

Nota: Obsérvese que las variables (AN_i).Test[$D_{i,z}$].Pasa o (AN_i).Test[$D_{i,z}$]. No-Pasa y (AN_i).Dispositivo[$D_{i,z}$] son dos variables interrelacionadas, de tal forma que la activación de la variable (de salida) (AN_i).Test[$D_{i,z}$].Pasa o (AN_i).Test[$D_{i,z}$]. No-Pasa indica que se hizo un test del dispositivo $_z$ (y lo paso o no lo paso), modificando así la variable (de entrada) (AN_i). Dispositivo[$D_{i,z}$], dejándole en el valor *Correcto* si paso el test o cambiándola al valor *Incorrecto* en el caso de no haberlo pasado.

Definición 11: Sea (AN_i).Tipo-Dispositivo[$D_{i,z}$]. Es una variable que indica el tipo de dispositivo que tiene el nodo respecto a su redundancia y tiene cuatro hechos asociados.

Crítico-No-redundante, Crítico-Redundante, S-Crítico y No-Crítico. Dispositivo crítico-No-Redundantes son los dispositivos conectados al N_i que no son redundantes, y además si fallan el nodo no puede funcionar mas (memorias, controladores de red y microcontrolador). Dispositivo S-Crítico es el dispositivo que no es redundante y que si falla no se le de baja al nodo del sistema (actuadores y/o sensores). Dispositivo No-crítico son los dispositivos redundantes en el nodo y que son los actuadores y/o sensores. Críticos-Redundantes son los dispositivos críticos antes mencionados pero se encuentran redundantes dentro del nodo i.

Donde:
(AN_i).Tipo-Dispositivo[$D_{i,z}$].Crítico-No-Redundante. Indica que el dispositivo $_z$ del nodo $_i$ es un dispositivo no redundante y además si falla al nodo i se le da de baja en el sistema.

(AN_i).Tipo-Dispositivo[$D_{i,z}$].Crítico-Redundante. Indica que el dispositivo $_z$ del nodo $_i$ es un dispositivo crítico y además redundante dentro del nodo i y si falla el nodo i no se le da de baja en el sistema, y AN_i recupera el fallo y los demás nodos, tareas y sistema no se enteran del fallo ocurrido dentro del nodo i.

(AN_i).Tipo-Dispositivo[$D_{i,z}$].S-Crítico. Indica que el dispositivo $_z$ del nodo $_i$ no es totalmente crítico y además no es redundante, el nodo sigue activo (un motor que no redundante o un sensor que no redundante).

(AN_i).Tipo-Dispositivo[$D_{i,z}$].No-Crítico. Indica que el dispositivo $_z$ del nodo $_i$ no es crítico porque existen en el nodo varios dispositivos iguales a él dentro del nodo y son los actuadores y/o sensores.

Definición 12: Sea *(AN_i).Todos-Dipositivos-con-fallos-Mismo-Tipo(z).* Es una variable que indica el estado en el sistema de un conjunto de actuadores y/o sensores redundantes, y tiene dos hechos asociados ***Alta y Baja***.

Donde:

(AN_i).Todos-Dispositivo-con-fallos-Mismo-Tipo(z).Alta. Indica que todos los dispositivos $_z$ del nodo redundante están dados de alta en el sistema.

(AN_i).Todos-Dispositivo-con-fallos-Mismo-Tipo(z).Baja. Indica que todos los dispositivos $_z$ del nodo redundante están dados de baja en el sistema.

4.6.2 Agente Tarea (AT_j)

Definición 13: Sea (AT_j).Fase. Es una variable que activa la distribución y estabilización de ***T***, de forma que puede obtener cinco hechos: ***Detección, Localización, Aislamiento, Reconfiguración y Recuperación***.
Donde:
(AT_j).Fase.*Detección*. En esta fase de detección de fallos se supone que la tarea se esta ejecutando correctamente.

(AT_j).Fase.*Localización*. Esta, fase se entra tras la detección de un error y se pretende localizar el elemento causante del mismo.

(AT_j).Fase.*Aislamiento*. En esta fase se intenta aislar a la tarea.

(AT_j).Fase.*Reconfiguración*. En esta fase se efectúa la reconfiguración de dicha tarea.

(AT_j).Fase.*Recuperación*. En esta fase se lleva a cabo la recuperación de la ejecución libre de errores de la tarea.

Definición 14: Sea (ATC_j).Fase. Es una variable que activa la distribución y estabilización de ***Tarea copia***, de forma que puede obtener cinco hechos: ***Detección, Localización, Aislamiento, Reconfiguración y Recuperación***.

(ATC_j).Fase.*Detección*. En esta fase de detección de fallos se supone que la **tarea copia** se esta ejecutando correctamente.

(ATC_j).Fase.*Localización*. Esta, fase se entra tras la detección de un error y se pretende localizar el elemento causante del mismo.

(ATC_j).Fase.*Aislamiento*. En esta fase se intenta aislar a la **tarea copia**.

(ATC_j).Fase.*Reconfiguració*. En esta fase se efectúa la reconfiguración de dicha **tarea copia**.

(ATC_j).Fase.*Recuperación*. En esta fase se lleva a cabo la recuperación de la ejecución libre de errores de la *tarea copia*.

Definición 15: Sea de forma que cuando mecanismos de detección de errores presentes en AT_j detectan un error, pasa esta variable a la situación *(AT_j).Entrada-Error.Error*.

Definición 16: Sea *(ATC_j).Entrada-Error*. Es una variable que se activa cuando la ATC_j detecta que la T_j en un determinado N_i no responde. Tiene dos hechos asociados:

Libre-de-Error* y *Error. En el inicio se supone que todas las entradas están ***Libre-de-Error,*** de forma que cuando mecanismos de detección de errores presentes en ATC_j detectan un error en AT_j, pasa esta variable a la situación *(ATCj).Entrada-Error.Error.*

Definición 17: Sea (ATj).Estado. Es una variable indica el estado de AT_j y tiene cuatro hechos: *Sospechoso, Correcto, Degradad y Baja*.

Donde:
(ATj).Estado.*Sospechoso*. Este estado indica que la T_j posiblemente tiene un fallo.

(ATj).Estado.*Correcto.* Este estado indica que el funcionamiento de la T_j está libre de fallos.
(ATj).Estado.*Degradad.* Este estado que la T_j está funcionando sin recibir o mandar la señal a todos los dispositivos de entrada o salida.
(ATj)Estado.*Baja.* Este estado indica que la T_j no funciona.

Definición 18: Sea (ATCj).Estado. Es una variable indica el estado de ATC_j y tiene cuatro hechos: *Correcto, Baja, activada y No-activada.*

Donde:
(ATCj).Estado.*Correcto.* Este estado indica que el funcionamiento de la TC_j está libre de fallos.

(ATCj).Estado.*Activad.* Este estado indica que la TC_j se activó.

(ATCj).Estado.No-*Activada*. Este estado indica que la TC_j se desactivó.

(ATCj).Estado.*Baja.* Este estado indica que la TC_j no funciona.

Definición 19: Sea (ATj).Sigue-Activa. Es una variable que indica que AT_j posiblemente logra reconfigurar a T_j, cuando falla una tarea de un nivel mas bajo (T_k,) esta relacionada con T_j. Tiene dos hechos asociados ***Activa y No-Activa***.

Donde:

(ATj).Sigue-Activa.Activa. T_j si se reconfiguro con otras tarea del sistema.

(ATj).Sigue-Activa.No-Activa. Indica que T_j no logro reconfigurarse con otras tareas y se debe dar de baja del sistema.

4.6.3 Agente Sistema (AS)

Definición 20: Sea (AS).Fase. Es una variable que activa la estabilización del AS que puede obtener cinco hechos: *Detección, Localización, Aislamiento, Reconfiguración y Recuperación.*

Donde:
(AS).Fase.*Detección*. En esta fase de detección se supone que *S* esta funcionando correctamente.

(AS).Fase *Localización.* En esta fase tras la detección de un error se pretende localizar el nodo o tarea causante del mismo.

(AS).Fase *Aislamiento-Nodo(i).* En esta fase se intenta aislar el Nodo fallido.

(AS).Fase *Aislamiento-Tarea(j).* En esta fase se intenta aislar el Tarea fallida.

(AS).Fase *Reconfiguración-Tarea(j).* En esta fase se efectúa la reconfiguración de la tarea j.

(AS).Fase *Reconfiguración-Nodo(i).* En esta fase se efectúa la reconfiguración de N_i.
(AS).Fase *Recuperación-Nodo(i).* En esta fase se lleva a cabo la recuperación del N_i.

(AS).Fase *Recuperación-Tarea(j).* En esta fase se lleva a cabo la recuperación del T_j.

Definición 21: Sea (AS).Estado. Es una variable indica el estado del N_i y tiene cuatro hechos: ***Sospechoso, Correcto, Degradado, Baja***.

Donde:

(AS).Estado.*Sospechoso(x*). Este estado indica que *AS*, posiblemente detecta un fallo en alguna tarea o nodo.

(AS).Estado.*Correcto*. Este estado indica un funcionamiento del *Sistema está* libre de fallos.

(AS).Estado.D*egradado(x)*. Este estado indica que en el *Sistema* existe algún Nodo o Tarea en fallo.

(AS).Estado.*Baja(x).* Este estado indica que él *Sistema* no funciona un nodo o una tarea(x).

Definición 22: Sea *(AS).Entrada-Error($_X$).* Es una variable que se activa por los diferentes mecanismos de detección de errores implementados en el AS, dependiendo si el error lo detecta en un N_i o en una T_j Tiene dos hechos asociados: ***Libre-de-Error*** **y** ***Error***. En el inicio se supone que todas las entradas están ***Libre-de-Error,*** de forma que cuando mecanismos de detección de errores presentes en AS detectan un error, pasa esta variable a la situación *(AS).Entrada-Error.Error ($_X$).*

Donde:

(AS).*Entrada-Error.Error ($_{Nodo}$).* Indica que un nodo ha fallado en el sistema.

(AS).*Entrada-Error.Error ($_{Tarea}$).* Indica que una tarea ha fallado en el sistema.

Definición 23: Sea (AS).Test[$_X$]. Es una variable es activada cuando se realiza un test a un nodo o una tarea con las acciones implementadas en el A*S* de realización de test para dicho componente del sistema. Y esta variable tiene 2 hechos asociados *Pasa y No-Pasa* .

Donde:
(AS).Test[$_X$].Pasa. Indica que el nodo o tarea x del sistema pasó el test que le aplicó el AS.

(AS).Test[$_X$].No-Pasa. Indica que el nodo o tarea x del sistema no pasó el test que le aplicó el AS.

(AS).Test[$_{Nodoi}$]. Pasa. Indica que el nodo i en el sistema pasó el test que le aplicó el AS.

(AS).Test[$_{Tareaj}$].Pasa. Indica que la tarea j en el sistema pasó el test que le aplicó el AS.

(AS).Test[$_{Nodoi}$]. No-Pasa. Indica que el nodo i en el sistema pasó el test que le aplicó el AS.

(AS).Test[$_{Tareaj}$].No-Pasa. Indica que la tarea j en el sistema pasó el test que le aplicó el AS.

4.6.4 Variables de comunicación entre agentes

4.6.4.1 Comunicación entre el Agente Nodo (AN_i) y el Agente Sistema (AS)

Definición 24: Sea (AN_iAS).Estado. Es una variable de comunicación entre (AN_i) y AS, y puede obtener tres hechos: ***Correcto, Baja y Degradado.***
Donde:
(AN_iAS).Estado.*Correcto.* En este estado el Agente N_i le indica al Agente Sistema que su funcionamiento es correcto.

(AN_iAS).Estado.Baja. En este estado el Agente N_i le indica al Agente sistema que esta en estado de baja.

(AN_iAS).Estado.*Degradado.* En este estado el Agente N_i le indica al Agente sistema que esta trabajando en estado degradado debido a que un Dispositivo tuvo un fallo permanente.

Definición 25: Sea (AN_iAS).Estado-Dipositivo[$D_{i,z}$]. Es una variable de comunicación entre el (AN_i) y (AS) con respecto al estado del dispositivo z dentro del N_i, y puede obtener dos hechos: ***Baja y Alta.***

Donde:
(AN_iAS).Estado. Dispositivo[$D_{i,z}$].Baja. En este estado el Agente Nodo le indica al Agente Sistema que el Dispositivo z esta dado de Baja.

(AN_iAS_j).Estado. Dispositivo[$D_{i,z}$].*Alta.* En este estado el Agente Nodo le indica al Agente Sistema que el Dispositivo z esta dado de Alta.

4.6.4.2 Comunicación entre el Agente Nodo (AN_i) y el Agente Tarea (AT_j)

Definición 26: Sea (AN_iAT_j).Estado.*A-Reconfigurar(z).*Es una variable de comunicación entre el (AN_i) y (AT_j).

Donde:
(AN_iAT_j).Estado.*A-Reconfiguar(z).* En este estado el Agente Nodo le indica al Agente Tarea que efectué su reconfiguración porque Dispositivo z dentro de N_i tiene un fallo permanentemente y existen más sensores o actuadores del mismo tipo que aún están activos por lo tanto la T_j se debe reconfigurar si puede.

Definición 27: Sea (AN_iAT_j).Estado-Dipositivo[$D_{i,z}$]. Es una variable de comunicación entre el (AN_i) y (AT_j), y puede obtener dos hechos: *Baja y Alta*.

Donde:
(AN_iAT_j).Estado. Dispositivo[$D_{i,z}$].*Baja.* En este estado el Agente Nodo le indica al Agente Tarea que el Dispositivo z esta dado de Baja.

(AN_iAT_j).Estado. Dispositivo[$D_{i,z}$].*Alta.* En este estado el Agente Nodo le indica al Agente Tarea que el Dispositivo z esta dado de Alta.

4.6.4.3 Comunicación entre el Agente Sistema (AS) y el Agente Nodo (AN_i)

Definición 28: Sea (ASAN$_i$).Estado. Es una variable de comunicación entre AS y (AN_i), y puede obtener dos hechos: ***Correcto, Degradado y Baja*** .

Donde:

(ASAN$_i$).Estado.*Correcto*. En este estado el Agente Sistema le indica al Agente N_i que su funcionamiento es correcto.

(ASAN$_i$).Estado.*Degradado*. En este estado el Agente sistema le indica al Agente N_i que esta trabajando en estado degradado porque están dados de baja otros nodos en el sistema.
(ASAN$_i$).Estado.*Baja*. En este estado el Agente sistema le indica al Agente N_i que esta dado de baja.

4.6.4.4 Comunicación entre el Agente Sistema(AS) y el Agente Tarea (AT_j)

Definición 29: Sea (ASAT$_j$).Estado. Es una variable de comunicación entre el (AS) y (AT_j), y tiene dos hechos ***Reconfigurada y No-Reconfigurada***.
Donde:
(ASAT$_j$).Estado.*Reconfigurada*. En este estado el Agente Sistema regresa el valor al Agente Tarea, de que la tarea ha sido reconfigurada.(cuando una tarea de un nivel más bajo que T_j falla da lugar a que T_j falle también, en el caso que AS pueda reconfigurar a la tarea j con otra tarea el sistema se lo comunica).

(ASAT$_j$).Estado.*No-Reconfigurada*. En este estado el Agente Sistema regresa el valor al Agente Tarea, de que la tarea no ha sido reconfigurada.(cuando una tarea de un nivel más bajo que T_j falla da lugar a que T_j falle también, en este caso AS no puedo reconfigurar a la tarea j con otra tarea el sistema se lo comunica).

4.6.4.5 Comunicación entre el Agente Tarea (AT_j) y el Agente Sistema (AS)

Definición 30: Sea ($AT_j A$S).Estado. Es una variable de comunicación entre (AT_j) y (AS), y puede obtener tres hechos: *A-Reconfigurar, Alta y Baja*.

Donde:
($AT_j A$S).Estado.*A-Reconfigurar*. En este estado el Agente Tarea le indica al Agente Sistema que la reconfigure con otra tarea porque la T_k(Tarea de mas bajo nivel que T_j) fallo.

($AT_j A$S).Estado.*Baja*. En este estado AT_j le avisa al AS que dio de baja a la T_j.
($AT_j A$S).Estado.*Alta*. En este estado el AT_j le avisa T_j esta dada de alta.

4.6.4.6 Comunicación entre el Agente Tarea (AT_j) y el Agente Nodo (AN_i)

Definición 31: Sea (AT$_j$AN$_i$).Estado.*Reconfigurada*. Es una variable de comunicación entre el (AT_j) y (AN$_i$), tiene dos hechos asociados Reconfigurada y No-Reconfigurada.

Donde:

(AT$_j$AN$_i$).Estado.*Reconfigurada*. En este estado el AT_j le indica al AN_i que si pudo reconfigurar a la T_j con los sensores o actuadores que aún están activos en el N_i.

(AT$_j$AN$_i$).Estado.*No-Reconfigurada.* En este estado el AT$_j$ le indica al AN$_i$ que no pudo reconfigurar a la T$_j$ con los sensores o actuadores que aún están activos en el N$_i$.

4.6.4.7 Comunicación entre el Agente Tarea (AT$_j$) y el Agente Tarea (AT$_k$)

Definición 32: Sea (AT$_j$AT$_k$).Estado. Es una variable de comunicación entre el (AT$_j$) y (AT$_k$), tiene dos hechos asociados Baja y Alta.

Donde:

(AT$_j$AT$_k$).Estado.*Baja.* En este estado el AT$_j$ le indica al AT$_k$ que dio de baja a la T$_j$.

(AT$_j$AT$_k$).Estado.*Alta.* En este estado el AT$_j$ le indica al AT$_k$ que dio de alta a la T$_j$.

4.6.4.8 Comunicación entre el Agente Tarea (AT$_k$) y el Agente Tarea (AT$_j$)

Definición 33: Sea (AT$_k$AT$_j$).Estado. Es una variable de comunicación entre el (AT$_k$) y (AT$_j$), tiene dos hechos asociados Alta y Baja.

Donde:

(AT$_k$AT$_j$).Estado.*Alta.* En este estado el AT$_k$ le indica al AT$_j$ que dio de alta a la T$_k$.

(AT$_k$AT$_j$).Estado.*Baja.* En este estado el AT$_k$ le indica al AT$_j$ que dio de baja a la T$_k$.

4.6.4.9 Comunicación entre el Agente Tarea Copia (ATC$_j$) y el Agente Sistema (AS)

Definición 34: Sea (ATC$_j$AS).Estado. Es una variable de comunicación entre el (ATC$_j$) y (AS), tiene dos hechos asociados Activada y No-Activada.

Donde:

(ATC$_j$AS).Estado.*Activada.* En este estado el ATC$_j$ le indica al AS que se activo.

(ATC$_j$AS).Estado.*No-Activda.* En este estado el ATC$_j$ le indica al AS que se desactivó.

4.6.4.10 Comunicación entre el Agente Tarea Copia (ATC$_j$) y el Agente Nodo (N$_i$)

Definición 35: Sea (ATC$_j$N$_i$).Estado. Es una variable de comunicación entre el (ATC$_j$) y (N$_i$), tiene dos hechos asociados Activada y No_Activada.

Donde:

(ATC$_j$N$_i$).Estado.*Activada.* En este estado el ATC$_j$ le indica al N$_i$ que se activo.

(ATC$_j$N$_i$).Estado.*No-Activada.* En este estado el ATC$_j$ le indica al N$_i$ que se desactivo.

4.6.4.11 Comunicación entre el Agente Tarea (AT$_j$) y el Agente Copia (ATC$_j$)

Definición 36: Sea (AT$_j$ ATC$_j$).Estado. Es una variable de comunicación entre el (AT$_j$) y (ATC$_j$), tiene dos hechos asociados Activada y No-Activada.

Donde:

(AT$_j$ATC$_i$).Estado.*Activada*. En este estado el AT$_j$ le indica al ATC$_j$ que se activo.

(AT$_j$ATC$_i$).Estado.*Baja*. En este estado el AT$_j$ le indica al ATC$_j$ que se desactivo.

4.6.4.12 Comunicación entre el Agente Tarea (ATC$_j$) y el Agente Tarea (AT$_j$)

Definición 37: Sea (ATC$_i$ AT$_j$).Estado. Es una variable de comunicación entre el (ATC$_j$) y (AT$_j$), tiene dos hechos asociados Activada y No-Activada.

Donde:

(ATC$_i$ AT$_j$).Estado.*Activa*. En este estado la ATC$_j$ le indica a AT$_j$ que se activo.

(ATC$_i$ AT_j).Estado.*No-Aactiva*. En este estado el ATC$_j$ le indica a AT$_j$ que se desactivo.

4.6.4.13 Comunicación entre el Agente Nodo (AN$_i$) y el Agente Tarea Copia (ATC$_j$)

Definición 38: Sea (AN$_i$$ATC_j$).Estado. Es una variable de comunicación entre el (AN$_i$), *y* (ATC$_j$) tiene dos hechos asociados Alta y Baja.

Donde:

(AN$_i$ ATC$_j$).Estado.*Alta*. En este estado la AN$_i$ le indica a ATC$_j$ que esta dado de alta.
(AN$_i$ ATC_j).Estado.*Baja*. En este estado el AN$_i$ le indica a ATC$_j$ que esta dado de baja.

4.6.4.14 Comunicación entre el Agente Sistema (AS) y el Agente Tarea Copia (ATC$_j$)

Definición 39: Sea (AS ATC$_j$).Estado. Es una variable de comunicación entre el (AS) y (ATC$_j$) (tiene dos hechos asociados Activada y No-Activada.

Donde:

(AS ATC$_j$).Estado.*Activada*. En este estado el AS le indica al ATC$_j$ que se activo.

(AS ATC$_j$).Estado.*No-Activda*. En este estado el AS le indica al ATC$_j$ que se desactivó.

4.7 Descripción esquemática de la arquitectura tolerante a fallos

Partiendo del esquema del agente reactivo con estado interno se tiene:

Agente nodo (AN$_i$): Será aquel agente que se encargara de monitorear el trabajo de los $[D_{i,z}]$ en N este trabajo será únicamente para cada nodo en forma independiente, y su labor es activar los mecanismos de detección, aislamiento y recuperación necesarios, estos mecanismos puede ser de dos maneras, hardware y software[Ors C.Rafael, Serrano Juan J., Alanis G.A, 2001],[Ors Carot, R., Alexandres G. G, 2002]. A Continuación se muestran los estados en los que pudiera estar (figura 4.6).

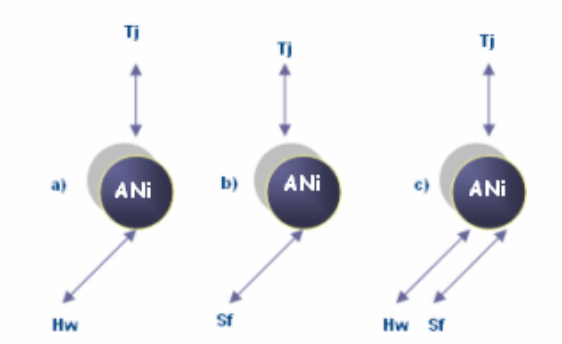


Figura 4.6 Agente nodo y su comunicación con el hardware y agente tarea.

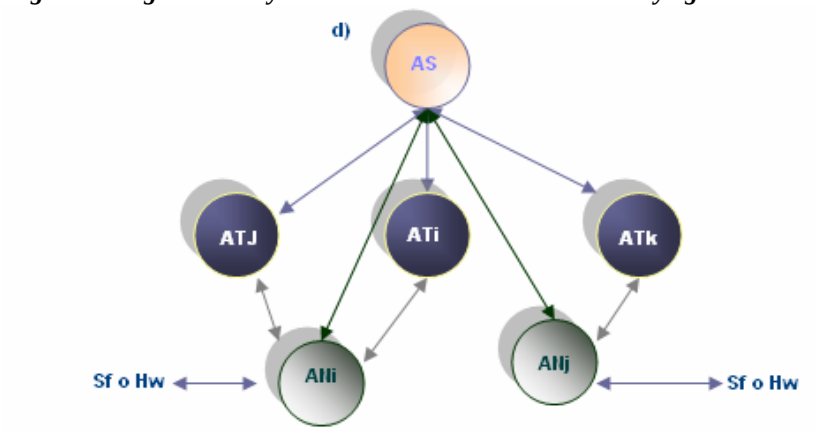


Figura 4.7 Comunicación del hardware con el agente nodo, comunicación entre el agente nodo con el agente tarea y con el agente sistema.

a) Agente Nodo(AN_i) con comunicación con hardware, además de con el Agente Tarea
b) Agente Nodo(AN_i) con comunicación con software, además de con el Agente Tarea
c) Agente Nodo(AN_i) con comunicación con software tanto con hardware, además de con el Agente Tarea
d) Agente Nodo(AN_i) con comunicación con el AS, además de con el Agente Tarea y con el Agente de otros nodos(figura 4.7) [Ors Carot, R., Alexandres G. G, 2001]

Agente tarea (AT_j): Es aquel agente que tiene como función monitorear y redistribuir las *Tareas* (T) en cada $[D_{i,z}]$, tendrá comunicación con el (AN_i), (AS) y (AT_k) (con otros agentes tareas).

A Continuación se muestran sus estados en la figura 4.8.

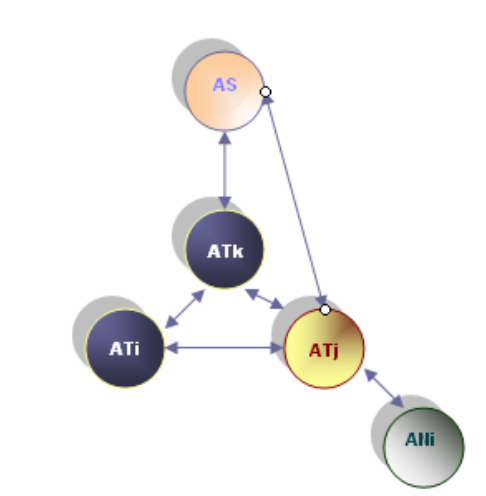


Figura 4.8 a) Agente tarea (At_j) con enlaces de comunicación con (AN_i),(AT_k) y (AS).

Agente sistema (AS): Será el agente que se encargue de monitorear todo el funcionamiento del *SMA*, teniendo comunicación con (AN_i) y (AT_j), e interactuando con el usuario. A continuación se muestra su estado (figura 4.9).

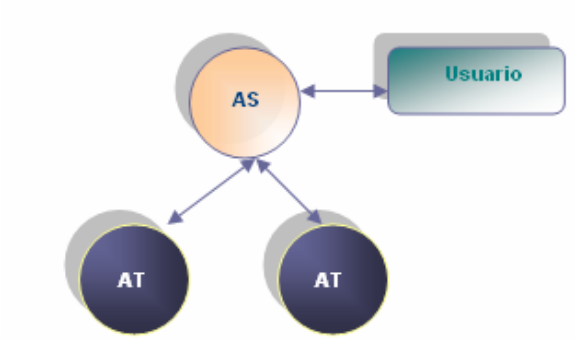


Figura 4.9 Agente sistema (AS) teniendo comunicación con (AT_j) y el usuario.

4.7.1 Comunicación y Cooperación

La comunicaron entre dichos agentes será en un forma ascendente y descendente dependiendo del tipo de agente (figura 4.10).

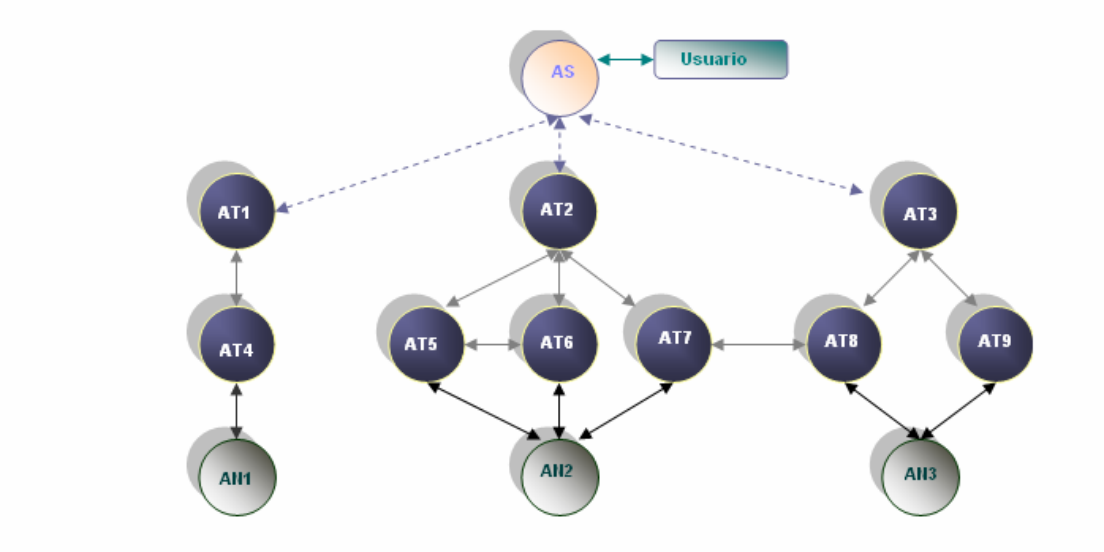


Figura 4.10 Comunicación entre los agentes vista esquemática

4.7.2 Ejemplos de arquitecturas de los sistemas de control en robots móviles

Ejemplo 1 sistema de control para un robot móvil reactivo sin integración de una capa de agentes tolerantes a fallos(figura 4.11).

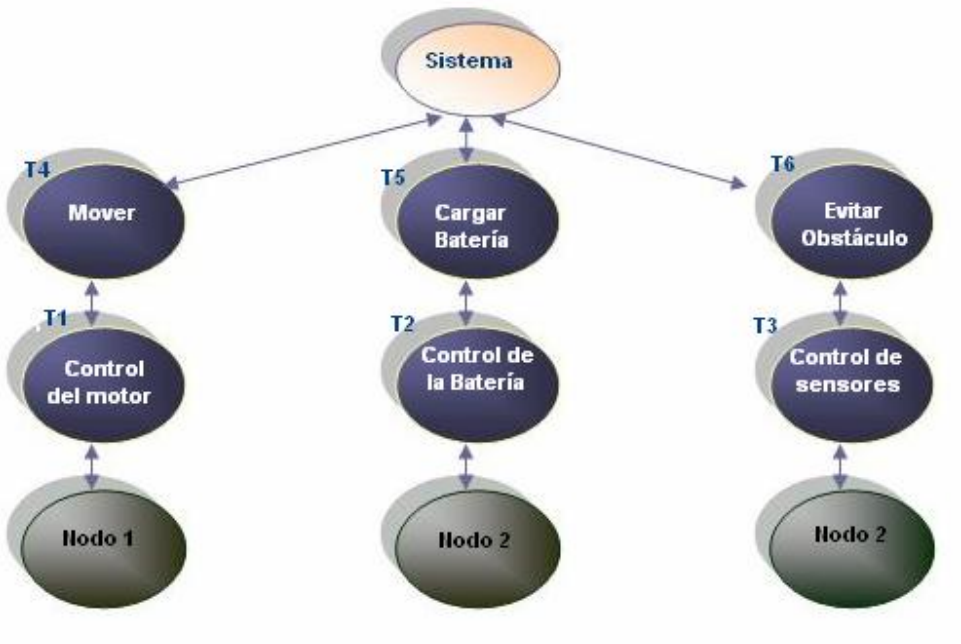


Figura 4. 11 Aquitectura de control reactiva en un robot móvil.

La figura 4.11, muestra el gráfico de una arquitectura de control reactiva para un robot móvil, en esta arquitectura no se tiene integrada la red de agentes tolerantes a fallos, por lo que si llegasen a tener un

fallo alguno de los componente que integran alguno de los nodos, el sistema dejará de funcionar, lo mismo sucederá si una de las tareas de primer o segundo nivel entran en modo de fallo [Ors Carot R, Serrano J. 2000]

Es la figura 4.12 se representa el ejemplo número 2, siendo este el gráfico del sistema de control reactivo para un robot móvil con la integración SMA tolerante a fallos. En este caso si uno de los componentes del sistema de control ya sea del software o hardware llegase a tener un fallo, los agentes que integran el SMA tolerante a fallos detectarán aislarán y reconfiguran el sistema para que este se mantega trabajando en la medida que le sea posible, obtenniendo así un sistema con mayor disposnibilidad que la arquitectura presentada en el ejemplo 1.

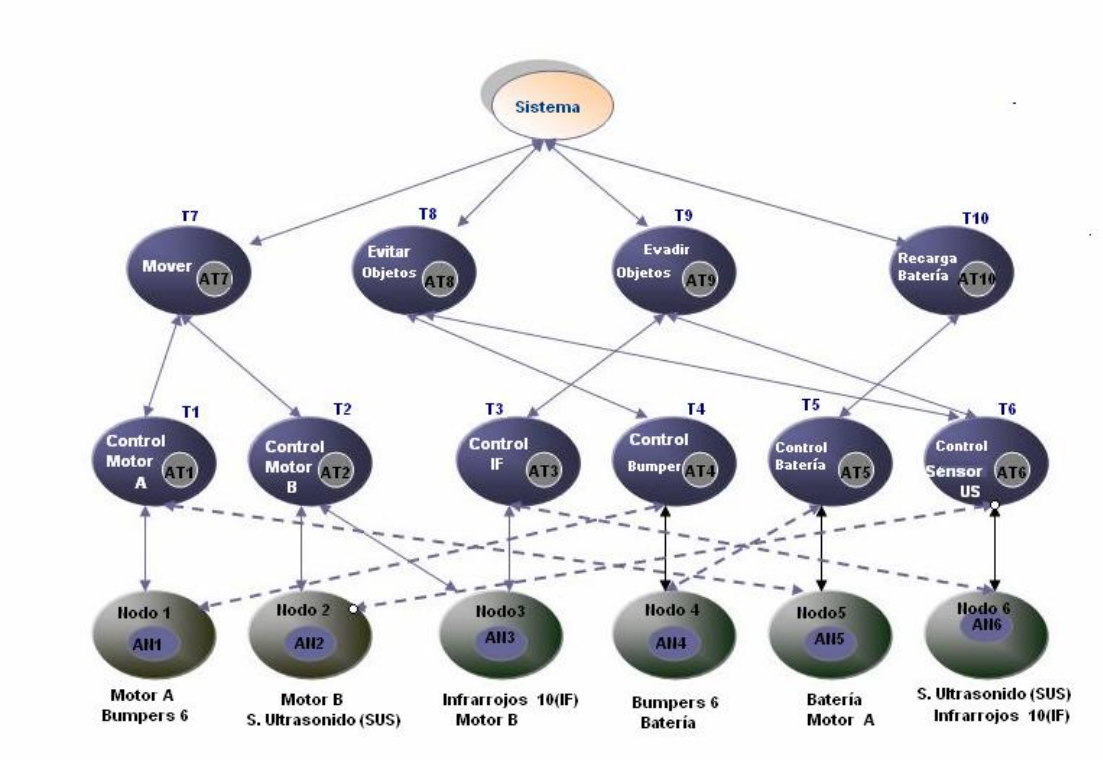


Figura 4.12 Arquitectura de control reactivo con la implementación del SMA tolerante a fallos.

En el *ejemplo 3* se prsenta el esquema en la figura 4.13 siendo este la arquitectura del sistema de control para un robot móvil (reactivo combinado con deliberativo) hibrído que tiene integrado el SMA tolerante a fallo. Con esto se puede puntualizar que el SMA tolernate a fallos es fácil de adaptar, sin tener que realizar grandes modificaciones al la red distribuida de agentes, en cualquiera de los tres tipos de arquitectura de control que existen para sistemas róboticos.

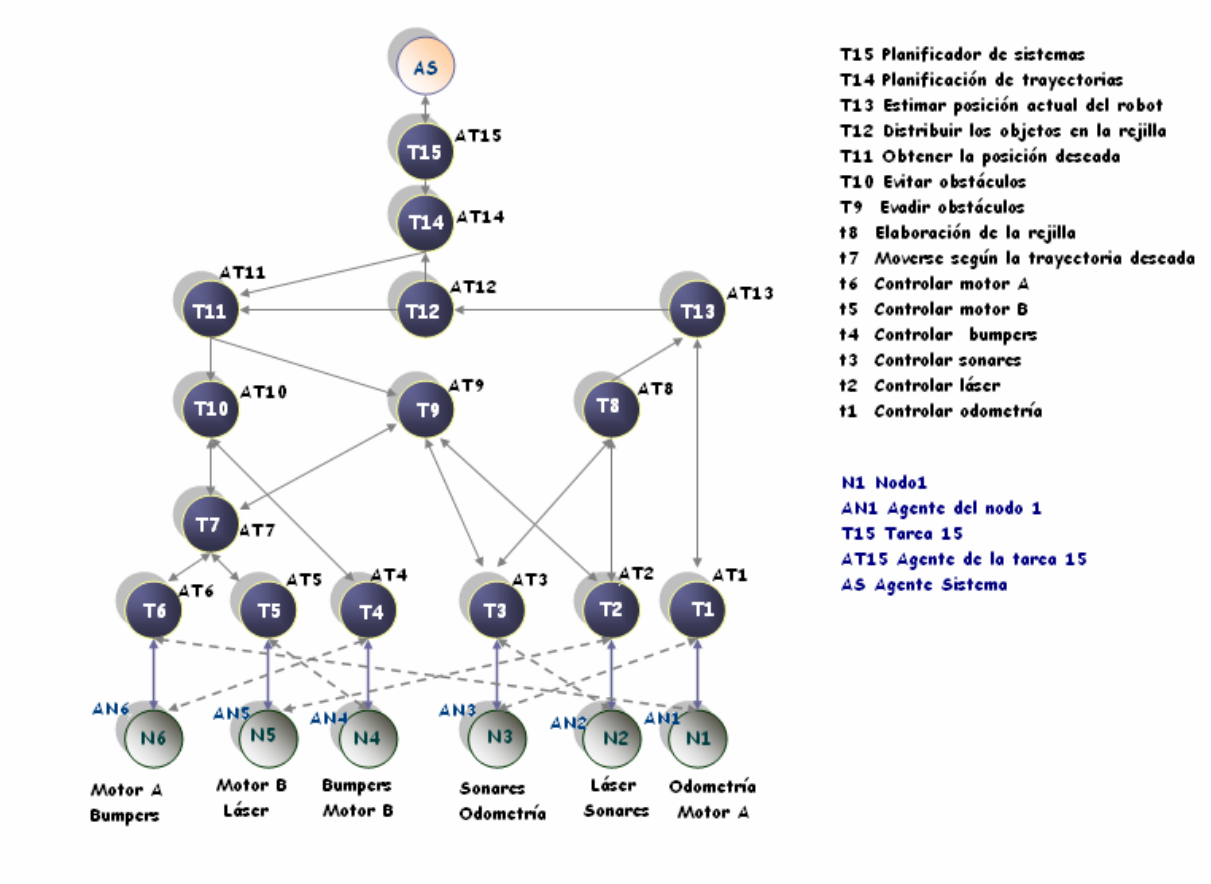


Figura 4.13 Arquitectura de control reactivo-deliberativo (híbrida) con implementación del SMA tolerante a fallos.

4.8 Metodología seleccionada para modelar el SMA tolerante de fallos

Ya que este sistema tiene un alto grado de dificultad, y la construcción de SMA integra tecnologías de distintas áreas como son:

- Técnicas de ingeniería del software para estructurar el proceso de desarrollo.
- Técnicas de inteligencia artificial para dotar a los programas de capacidad para tratar situaciones imprevistas y tomar decisiones.
- Programación concurrente y distribuida para tratar la coordinación de tareas ejecutadas en diferentes computadoras bajo diferentes políticas de planificación.
- Técnicas adecuadas de comunicación entre los agentes.

Debido a esta combinación de tecnologías, el desarrollo de SMA se complica. Y por tal motivo es imperante utilizar una metodología de desarrollo de software especializado donde se realice el análisis y diseño de una manera ordenada y formal.

Después de analizar la estructura de metodologías expuestas anteriormente se llegó a la conclusión que para desarrollar un SMA de una manera formal en lo que respecta al análisis y diseño se debe de tomar en cuenta los siguientes conceptos de la metodología y estos son:

- Especificación de planificación de tareas.
- Modelado de tareas concurrentes
- Intercambio de información con lenguajes de comunicación orientados a agentes.
- Motivación de los componentes del sistema.
- Diseño formal de conversaciones entre los diversos agentes que integran el sistema.
- Validación de las conversaciones.
- Implementación de la inteligencia.
- Estructura interna del agente.
- Estructura software para Sistema Multiagente.

Tomando en cuenta los puntos anteriores se decidió utilizar la metodología denominada MaSE ya que cuenta con las siguientes características adecuadas para realizar el análisis y diseño del SMA propuesto en este trabajo.

MaSE es una metodología desarrollada en la Fuerza Aérea de los Estados Unidos. Se diseñó específicamente para modelar Sistemas Multiagente heterogéneos. Se contempla a los agentes como entidades que no necesariamente deben tener inteligencia y que surgen como una especialización de los objetos, por lo cual, las técnicas que utiliza MaSE provienen del paradigma orientado a objetos. MaSE abarca desde la especificación inicial del sistema (*provista por el usuario*) hasta la implementación del Sistema Multiagente. Se basa en el concepto de roles mediante los cuales se definen las clases de agente a implementar. En el análisis, se definen los roles y en el diseño se definen las clases de agente y sus interacciones. La metodología es iterativa, lo cual facilita el refinamiento tanto del análisis como del diseño. Además se pueden volver pasos hacia atrás tantas veces como sea necesario (*no es una metodología estrictamente secuencial y progresiva*).

MaSE como se mencionó anteriormente se basa en el paradigma orientado a objetos y asume que un agente es una especialización de un objeto. La especialización consiste en que los agentes se coordinan unos con otros por medio de conversaciones y actúan proactivamente para alcanzar metas individuales, y de esta manera poder lograr que el sistema cumpla con un objetivo global.

En la metodología MaSE los agentes son sólo una abstracción conveniente, que puede o no poseer inteligencia. En este sentido, los componentes inteligentes y no inteligentes se gestionan igualmente dentro de la misma plantilla.

La metodología MaSE es independiente de la arquitectura de sistema, del lenguaje de programación, o del sistema de comunicación en particular. Un MAS diseñado con la metodología MaSE puede implementarse de varias maneras a partir del mismo diseño. Los desarrolladores de MaSE prevén que en un futuro en la herramienta de desarrollo denominada AgentTool deberán de implementase perfectamente esta independencia, permitiendo que el diseño del MAS produzca código en varios lenguajes incluyendo C++ y Java.

Una fortaleza de la metodología MaSE es su habilidad para dar seguimiento a los cambios a través del proceso. Cada diseño se puede rastrear hacia atrás o hacia delante a través de las diferentes fases de la metodología y sus correspondientes conceptos. De esta manera, se puede dar seguimiento a la inversa para encontrar los requerimientos iniciales que apoyan a un agente en particular. Además se puede hacer lo opuesto, un agente de una fase inicial como una meta se puede mapear a un conjunto de agentes de fase posterior. El propósito es poder seleccionar un objeto de diseño en la herramienta de desarrollo AgentTool y recibir una retroalimentación visual de todos los demás objetos y sus efectos. A continuación se enlistan sus ventajas y desventajas:

Ventajas

- Presenta los pasos claramente definidos por lo cual su aplicación no representa dificultades.
- Se basan en metas las que casi no cambian durante el desarrollo del sistema a diferencia de las metodologías basadas en tareas y actividades.
- Tanto análisis y el diseño giran en torno al concepto central de rol, lo cual hace de MaSE una metodología fácil de entender.
- Como primer paso analiza los objetivos del sistema, permitiendo, por un lado dejar bien en claro a qué apunta el sistema. Y por otro lado, utiliza un modelo y conceptos que son fáciles de entender por el usuario.
- Utiliza ciertos modelos que son propios de Unified Modeling Lenguaje (ULM), como los Casos de Uso (metas en MaSE), diagramas de secuencias, diagramas de tareas, diagramas de comunicación y diagramas componentes. Con esto, no se requiere aprender demasiados modelos nuevos.
- Los modelos propios de MaSE son fácilmente interpretables.
- Cuenta con una herramienta propia de desarrollo, AgentTool, la cual genera código en forma automática, a partir de los modelos del análisis y del diseño.
- AgentTool permite verificar cuestiones como la consistencia de modelos entre análisis y diseño.

Desventajas

- No contempla relaciones genéricas entre clases de agente estereotipadas.
- El hecho de contar con una herramienta de desarrollo específica hace que la etapa de implementación de MaSE esté dentro de dicha herramienta y no en forma explícita en la metodología.
- No parece factible elegir cuál herramienta de desarrollo se desea utilizar, siendo obligatorio el uso de AgentTool (si no posee las características deseadas o el soporte adecuado, habrá que resignarse a esta).
- El SMA que se desarrolla con esta metodología es un sistema cerrado.
- El SMA desarrollado con esta metodología soporta cuando mucho 10 clase diferentes de agentes, ya que no existe validación y verificación en AgentTool para sistemas de gran magnitud.

Mejoras que se están realizando actualmente.

- Traducir todos los modelos propios de MaSE a modelos UML, para seguir un estándar de modelado.
- Obtener una descripción del diseño que cuente con relaciones estereotipadas de clases de agentes, mediante UML.
- Definir un nexo entre el diseño y la implementación, que resulten de un proceso independiente de cualquier herramienta de desarrollo.
- Ampliar el número de clases de agentes.

Cuatro ventajas con las que cuenta esta metodología fueron las que se tomaron principalmente en cuenta, para seleccionar esta metodología en el desarrollo del modelo MAS tolerante a fallos:

La primera ventaja tomada en cuenta es que MaSE se enfoca hacia las habilidades específicas de los Sistema Multiagente, basándose en regalas y un lenguaje formal. La metodología soporta Sistemas Multiagente distribuidos ayudando de cierta manera en nuestro trabajo al diseño de estructuras de comunicación entre los agentes que requerimos para realizar la configuración del sistema ante un

determinado fallo en alguno de sus componentes. Esto es posible porque la metodología esta enfocada en el dominio de los agentes, lo que significa que al modelar nuestra propuesta estamos usando una metodología enfocada totalmente al concepto que se tiene del paradigma basado en agentes.

La segunda ventaja es que MaSE soporta el modelado del SMA en su herramienta desarrollado (AgentTools) el cual esta basado sobre un lenguaje gráfico contando el diseñador del sistema con una interfase gráfica, con la finalidad de que todas las fases de la metodología se desarrollen sobre agentTool. De tal manera los diagrama gráficos nos dan una información visual de la dimensión del sistema que se está desarrollando, ocultando a los constructores formales.

La tercera ventaja es la validación y verificación de las tareas y las conversaciones entre las diferentes clases de agentes, así como la representación mediante autómatas de estado finito de tareas, conversaciones y componentes.

La cuarta ventaja es que ofrece un método de transformación y especificación del sistema pasando de la fase de análisis al la fase diseño, en la mayoría de las metodologías aquí mencionadas carecen de esta dirección específica.

4.9 Arquitectura software seleccionada para interconectar el SMA tolerante a fallos

Una estructura software para el control de un robot móvil trata de agrupar a los procesamientos de sensores, actuadores, y los algoritmos de navegación en un sistema funcional previendo una estructura de procesamiento. Esta sección se iniciará con la introducción de algunos conceptos y trabajos previos relacionados con la arquitectura de control en robots.

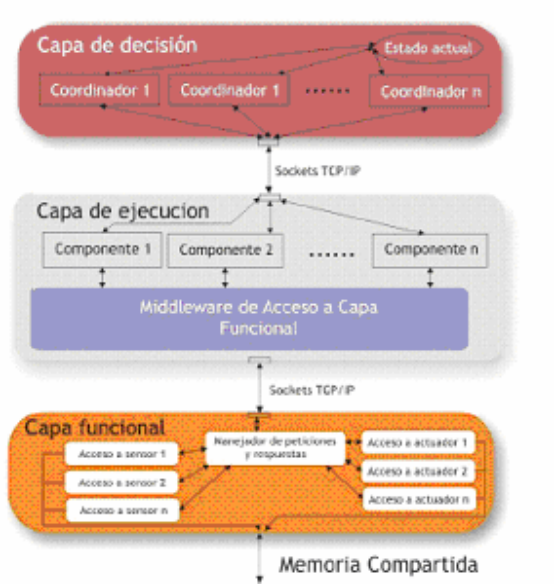
La funcionalidad y validez de un sistema para que pueda ser considerado como un enfoque orientado hacía la robótica depende de ciertos factores, entre los más importantes destacan [Félix 2003]:

Integración: Que se establezcan mecanismos transparentes de comunicación entre los componentes.
Reactividad: Que los componentes que integran la arquitectura sean capaces de reaccionar apropiadamente ante estímulos específicos y situaciones particulares tomando en consideración los requerimientos de las aplicaciones en tiempo real.
Robustez y tolerancia a fallas: La capacidad de explotar la redundancia de recursos, fuentes de información y procesos, son tareas que la arquitectura deberá permitir, además de contar con un sistema de tolerancia a fallas, que permita recuperar la funcionalidad del robot en entornos semi-controlados.
Seguridad: Se requieren mecanismos que garanticen la seguridad de las aplicaciones robóticas, puesto que algunas veces, el robot puede verse implicado en situaciones donde la seguridad humana es un factor muy importante asimismo desde el punto de vista de garantizar la integridad del robot.
Extensibilidad: La arquitectura debe permitir agregar nuevas funcionalidades que puedan ser fácilmente adaptables a las existentes.
Distribución de tareas: poder emplear un sistema distribuido o centralizado según sean las necesidades.

4.9.1 Arquitectura software 3+

Esta arquitectura fue diseñada por el M.C. José Antonio Pacheco Sánchez en su trabajo de tesis de maestría en Ciencias computacionales dentro del gripo de investigación de Inteligencia Artificial en el Instituto Tecnológico de monterrey ITESM Campus Morelos bajo la dirección del Dr. Luis Enrique Surcar Surccar.

Este modelo arquitectónico está basado en capas denominado 3+ (se basa en un esquema de arquitectura híbrida) bajo un ambiente de cómputo distribuido. Las capas que forman parte de la arquitectura son: *la capa funcional*, o de nivel más bajo, misma que interactúa con los dispositivos físicos del robot, *la capa de ejecución*, donde se localizan los componentes de movilidad e interfaz humano-robot que integran la arquitectura y la *capa de decisión*, misma que se localiza en el nivel superior de esta arquitectura y realiza tareas de coordinación. En la figura 4.14 se muestra un panorama general que describe esta arquitectura.



ROBOT

Figura 4.14 Diagrama de la arquitectura 3+.

4.9.1.1 Capa funcional

Esta capa es la encargada de manejar el control reactivo del robot interactuando directamente con los sensores y actuadores a través de interfaces genéricas para dispositivos. Localizada en el nivel más bajo de la arquitectura, incluye todas las capacidades integradas con los sensores y actuadores del robot. Estas funciones de procesamiento y ciclos de control (procesamiento de imágenes de bajo nivel, control de movimiento, etc.) se encuentran encapsulados en componentes comunicados entre sí.

4.9.1.2 Capa de ejecución

Localizada en medio de las capas de la arquitectura 3+, esta capa consiste en una serie de componentes individuales dedicados al procesamiento específico de una tarea. Esta capa incluye los componentes que conforman la arquitectura. Por ejemplo los componentes de movilidad (navegación, localización y planeación de trayectorias) y de interacción entre el ser humano y el robot (Interfaz con asistente personal digital, procesamiento de voz, detección de personas, reconocimiento de rostros y reconocimiento de ademanes).

En esta capa, se encuentra también una sub-capa de *middleware* que permite hacer una abstracción de las interfaces que se utilizan para la comunicación semántica entre los componentes que integran la capa de ejecución y los componentes que se sitúan en el nivel funcional. Los mensajes generados desde

los componentes se envían hacia la capa de decisión del sistema donde los coordinadores se encargan de recibir y reenviar mensajes entre los módulos, generar y actualizar el vector que representa el estado del mundo con base en los mensajes recibidos desde los componentes del nivel de ejecución y enviar la solicitud de una determinada acción al componente que corresponda.

4.9.1.3 Capa de decisión

La capa de decisión es la capa de alto nivel dentro del contexto de la arquitectura 3+. En esta capa existen coordinadores que llevan a cabo las funciones de actualizar las variables de ambiente y de enviar a los componentes correspondientes el mensaje que desencadenará la acción de acuerdo al modelo actual. Los coordinadores conocen y establecen los mecanismos a través de los cuales los elementos que componen la arquitectura son enlazados entre si para llevar a cabo un plan de acuerdo a una tarea global.

La capa de decisión tiene la política de tomar decisiones que están dadas en términos de los valores de las diferentes variables de estado. Actualmente se cuenta con una tabla de relación estado/acción que el coordinador utiliza de manera determinística para seleccionar la acción de acuerdo al conjunto de valores que conforman el modelo de estado. De acuerdo a los componentes que se vayan agregando a la capa de ejecución se pueden considerar un mayor número de acciones que el robot puede llevar a cabo. Lo único que se tiene que hacer, es extender el modelo de estado agregando las variables que el componente defina y generar las políticas adecuadas para el nuevo conjunto de variables.

La arquitectura 3+ integra un mecanismo de estados ligados a un conjunto de acciones que el robot requiere ejecutar para llevar a cabo una meta o plan de acuerdo a una tarea global. Estos estados son controlados por uno o más coordinadores localizados en la capa de decisión, mismos que se valen de un modelo del mundo, generado a partir de entradas recibidas desde los componentes localizados en la capa de ejecución. Estos coordinadores se comunican con el nivel de ejecución a través de mensajes con la finalidad de ejecutar la acción correspondiente en el componente indicado por el coordinador.

Uno de los aspectos más importantes en la definición de la arquitectura 3+, es el carácter distribuido entre los componentes localizados sobre la capa de ejecución y en la interacción de éstos con el nivel de decisión. Al definir estos componentes como unidades independientes de procesamiento que en su conjunto dan forma a la arquitectura 3+, no necesariamente tienen que estar ejecutándose sobre el mismo equipo que se encuentra localizado en el cuerpo del robot. Se estableció un protocolo o esquema de comunicación para la comunicación entre componentes. Este protocolo marca la pauta para que las aplicaciones que se requieran integrar a la arquitectura, cumpliendo con los requisitos que el protocolo de comunicación establece.

4.9.1.4 Ventajas de la arquitectura 3+

Esta arquitectura 3+ nace de la necesidad de cómputo demandante que requieren ciertas aplicaciones, como el procesamiento de imágenes y reconocimiento de voz. Con el diseño de esta arquitectura se quiere obtener un robot *ligero*, de tal manera que las aplicaciones puedan poner a disposición en servidores remotos con gran capacidad de carga, ya sea en el procesamiento de información, como de las comunicaciones. Los componentes de la arquitectura fueron analizados para su posible implementación bajo un esquema de servicios robóticos. Esto quiere decir, que algunos componentes se pueden implementar a través de servidores en una red, mismos que recibirán peticiones desde las entidades robóticas de acuerdo a las acciones que el robot necesite ejecutar para cumplir una meta asignada por el usuario, estas serán analizadas y ejecutadas de acuerdo a las restricciones del servicio y posibilidades en cuanto a recursos se refiere.

Una vez que es ejecutada la petición, se generará una respuesta que es enviada a su origen. Para este modelo de arquitectura se requirió definir:

- Un protocolo de comunicación basado en cadenas de texto para intercambiar datos entre los componentes de la arquitectura. Dicho protocolo está estructurado en unidades de datos (UD) compuestas por un encabezado, una sección de datos y un terminador de trama que sirve para comprobar la integridad del mismo.
- Los modelos de enlace de comunicaciones alambicas e inalámbricas que ofrezcan la robustez necesaria para soportar de manera adecuada la arquitectura.
- Especificaciones de requerimientos en tiempo real que deberán cubrir las aplicaciones que se convertirán en servicios.
- Un modelo de arquitectura distribuida que soporte la interacción entre los servicios y las entidades robóticas.

La arquitectura 3+ establece que cualquier componente dentro de la capa de ejecución puede ejecutarse en un equipo diferente y comunicarse con el coordinador en la capa de decisión que a su vez también puede estar localizado en cualquier otro equipo. Esta comunicación se realiza a través de sockets TCP/IP, lo cual garantiza que cualquier componente, actual o futuro, puede integrarse de una manera rápida y sencilla a la arquitectura ya que sólo necesita tener una conexión a la red en la cual se manejan los procesos y servicios robóticos y al haber establecido previamente su protocolo de comunicación de paso de mensajes. Es a través de este esquema, como se soluciona el problema de la necesidad de varias aplicaciones de tener gran capacidad de cómputo, ya que algunos componentes, por ejemplo el manejo de imágenes en tiempo real, requieren de gran capacidad de procesamiento [Pacheco Sánchez José A. 2005].

4.9.1.5 Protocolo de comunicación

Uno de los principales retos de una arquitectura de software, es el de unir los desarrollos y aplicaciones independientes y convertirlas en componentes, que ejecutarán tareas específicas para llevar a cabo una meta global. Estas aplicaciones son autónomas en cuanto a plataforma, lenguaje y metodología de desarrollo. Por ejemplo, algunas de ellas están elaboradas en plataformas UNIX, como LINUX utilizando Java como lenguaje de desarrollo y otras en plataformas Windows utilizando .NET como ambiente de desarrollo.

Esta arquitectura propone la creación del concepto de Servicios Robóticos. Este concepto establece un protocolo de comunicación para el paso de mensajes entre cada uno de los componentes que integran la arquitectura. Los componentes que forman parte del sistema, deben de contar con un esquema de comunicación basado en el protocolo TPC/IP, mismo que permite una fácil implementación para el envío de mensajes a través de tramas que contienen la información de entradas y salidas para cada uno de los servicios robóticos. La decisión que se tomó de implementar el esquema de comunicación basado en mensajes a través de TCP/IP se debe a la sencillez de programación que provee. Para la arquitectura tolerante a fallos propuesta en este trabajo, los componentes que interactúan en ella son tratados como *cajas negras* en el sentido de que las funciones y la programación se encuentran encapsuladas en dichos componentes. Los mensajes que se definen para cada componente se establecen como interfaces para acceder a tales funciones.

4.9.1.6 Estructura del protocolo de comunicación

El protocolo de comunicación define un esqueleto para la representación de datos que cada componente utiliza. La estructura de un mensaje se encuentra establecida en la figura 4.15 mostrando un espacio

para definir el encabezado, un espacio para los datos de intercambio entre componentes y el espacio final para la implementación de un sistema verificador de la integridad de la información.

Figura 4.15 Estructura de un mensaje del protocolo de comunicación.

El encabezado es el mismo para todos los mensajes, independientemente del componente que lo envía o recibe. A través del encabezado, el coordinador determina el equipo origen, el componente origen, y el tipo de mensaje que se desea enviar y realiza las operaciones correspondientes. Por ejemplo, si se desea manipular el robot a control remoto, el componente PDA tiene que enviar un mensaje al coordinador solicitando la operación del robot en modo de control remoto. El coordinador verificará el estado del modelo y actualizará la variable correspondiente. Posteriormente, reenviará el mensaje hacia el equipo y componente que el encabezado muestre.

En la tabla 4.9, se muestra el contenido del encabezado para el protocolo de comunicación. Este encabezado contiene la información necesaria para identificar el origen y el destino del mismo. Por ejemplo, en el caso de la tabla 4.9 se está enviando el paquete con identificador *001* a las 15:25:10 en el formato hora, minutos, segundos (hh mm ss) desde el equipo *PDA01* que está ejecutando el componente *PDA* el cual envía el mensaje identificado como *POS* (solicita la posición actual del robot). Este mensaje tiene como destinatario el equipo SERVPLAN que ejecuta el componente PLA (Planeador). Como parte el encabezado, se incluye también un lugar para el código de error, mismo que será llenado por el cliente cuando se detecte un error dentro del procedimiento determinado [Pacheco Sánchez José A. 2005].
.

Header						
Propiedad	*Longitud*	*Posición inicial*	*Posición final*	*Descripción*	*¿Quién lo llena?*	*Dato ejemplo*
IDPAQUETE	*3*	*1*	*3*		*cliente*	*001*
HORAEMISION	*6*	*4*	*9*		*cliente*	*152510*
IDORIGEN	*8*	*10*	*17*		*cliente*	*PDA01*
COMORI	*8*	*18*	*25*		*cliente*	*PDA*
PROPIEDAD	*8*	*26*	*33*		*cliente*	*POS*
IDDESTINO	*8*	*34*	*41*		*cliente*	*SERVPLAN*
COMDES	*8*	*42*	*49*		*cliente*	*PLA*
CODDER	*3*	*50*	*52*		*cliente*	*100*

Tabla 4.9 Encabezado del protocolo de comunicación.

4.9.1.7 Interfase entre la capa tolerante a fallos y la arquitectura software 3+

Se puede concluir que independientemente de la arquitectura software de control del robot, se puede implementar una pequeña interfase con el Sistema Multiagente tolerante a fallos sin mucha

complicación. En el caso particular planteamos que nuestro Sistema Multiagente tolerancia a fallos se puede interconectar con la arquitectura 3+ usando como interfase un coordinador que sirve como interfaz a la capa tolerante a fallos en el nivel de decisión

4.10 Diseño de la interfase entre el SMA tolerante a de fallos propuesto y la arquitectura software 3+

La arquitectura 3+, se modifico en este trabajo agregando un coordinador dentro del nivel de decisión con la finalidad de manejar el aspecto de tolerancia a fallos. Este coordinador es el encargado de incluir en una estructura de datos (árbol de búsqueda binaria dinámico) todos los sensores y actuadores, en el lado izquierda del árbol se localizan los sensores y en lado derecho todos los actuadores, cada nodo de dicho árbol contiene el tipo de sensor, el nodo donde se encuentra conectado de manera activa, en que nodo esta su doble conexión, y si esta replicado, en caso que tenga replica se describe el nodo donde se localiza dicha réplica, de tal manera que el *Agente Sistema Activo* la pueda consultar, y de esta manera conocer los tipos y cantidad de lo dispositivos de entrada y salida que tiene el robot, y así poder generar y dar de alta a los *Agentes Nodos* necesarios en cada nodo que conforma el sistema de control, la información antes mencionada se transfiere al árbol desde una base de datos, y se almacena en otra base de datos exclusiva del agente sistema, todo este mecanismo de transferencia de información lo realiza el nuevo coordinador que sirve de interfaz, entre la arquitectura 3+, y el SMA tolerante a fallos que conforma la arquitectura propuesta..

En el caso de las tareas el mismo coordinador generará una lista doblemente ligada dinámica la que almacenará cada una de las tareas que conforma el sistema de control del robot, en que nodo se localiza, el número de copias que tiene, tipo de tarea, etc. estas serán almacenadas en una memoria tipo pizarra para que sean consultada por el *Agente Sistema Activo*, y de esta manera este generará y dará alta a los agentes tarea necesarios. La información antes mencionada también es obtenida de una base de datos y se transfieren a la lista doblemente ligada en la pizarra, y esta a su vez es transferida a la base de datos de agente sistema.

En la figura 4.16 se muestra la capa de decisión modificada con la interfase siendo esta el coordinador y sus estructuras de datos así como la memoria compartida, y el Agente Sistema Activo de esta manera se puede interconectar a la estructura software 3+ el Sistema Multiagente tolerante a fallos. La base de datos del *Agente Sistema Activo* almacenará también todos los fallos que presenten los diferentes componentes de software y hardware del sistema robótico.

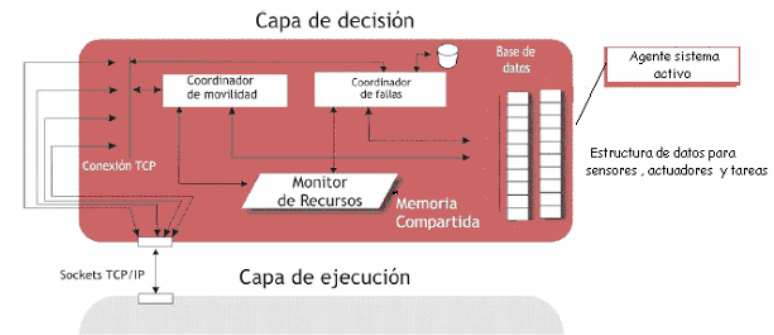


Figura 4.16 La capa de decisión en la arquitectura 3+ con su interfase para tolerar fallos mediante el SMA propuesto en este trabajo.

En la fase de implementación de la capa tolerante a fallos en la arquitectura 3+, la política será generada por el usuario de forma manual, siendo una política determinística.

4.10.1 Tolerancia a fallos a nivel comportamiento

En la arquitectura 3+, se diseño la inclusión de un nuevo coordinador dentro del nivel de decisión con la finalidad de manejar el aspecto de tolerancia a fallos a nivel de los comportamientos del robot. Dicho coordinador se le encarga administrar una base de datos de fallos de comportamiento del robot, la cual tiene la finalidad de almacenar la información de los fallos que cada componente pueda tener a lo largo de la operación del robot, de tal manera que a medida que se vayan agregando componentes que definen comportamientos a la arquitectura, esta base de datos cuenta con la información específica de las fallos que pueden ocurrir en dicho componente.

En esta base de datos, se almacenan los posibles valores de fallo de cada uno de los componentes. También se cuenta con otra la base de datos donde se almacenan los estados no válidos los cuales pueden ser detectan del modelo del mundo. El coordinador de fallo es capaz de detectar cuándo el valor de las variables del modelo del mundo representa un estado no válido de acuerdo a su base de datos. Por lo que se propone una especificación de los mecanismos de tolerancia a fallos en base a lo anteriormente expuesto (coordinador de fallos y mecanismo de base de datos de fallos, mensajes y recuperación de fallos).

En la figura 4.17 se denota el comportamiento de las acciones y las variables, a través de un diagrama de estados de UML definidos para la arquitectura, se propone que el inicio se realice en el estado de espera.

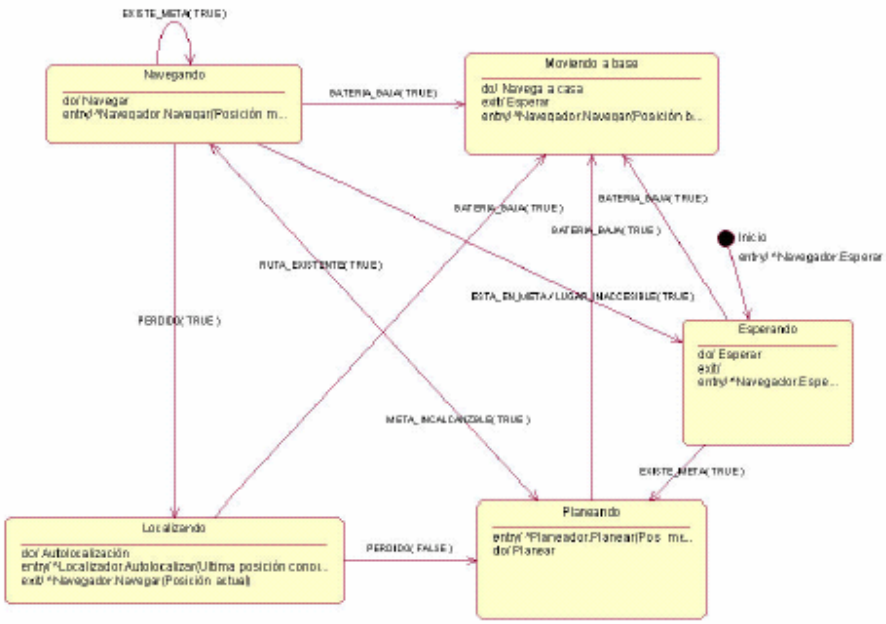


Figura 4.17 diagrama de estados UML para la arquitectura 3+.

En la Tabla 4.10 se muestra una lista de las variables de estado para el ejemplo de la arquitectura y los posibles valores que pueden tener.

	Variable	Descripción	Quién modifica	valores
1	**EXISTE_META**	***Obtiene la posición de la meta***	***PDA***	***Bool***
2	**ESTA_EN_META**	***Verifica si el robot está en la posición meta***	***Localizador***	***Bool***
3	**LUGAR_INACCESIBLE**	***Si el robot no puede llegar a la meta , pasado 3 intentos de replantear la trayectoria esta variable toma el valor de Verdadero***	***Navegador***	***Bool***
4	**BATERÍA_BAJA**	***Se activa cuando la batería está baja, el valor Verdadero de ésta variable, por sí sola, desencadena la acción a que el robot se dirija al laboratorio***	***Monitor de recursos***	***Bool***
5	**PERDIDO**	***Cuando el robot pierde la certidumbre con respecto a su localización, ésta variable se activa en Verdadero***	***Navegador***	***Bool***
6	**CONTROL _ REMOTO**	***Se activa cuando se accede al robot a través de la PDA, en su modalidad de tiempo real***	***PDA***	***Bool***
7	**META_INALCAZABLE**	***Se activa cuando una meta no es alcanzada, pero se tiene la posibilidad de replantear su trayectoria hasta 3 intentos, Si después de estos 3 intentos aún no se logra un plan válido para llegar a la meta, se activa la variable LUGAR_INACCESIBLE***	***navegador***	***Bool***
8	**RUTA_EXISTENTE**	***Su valor es Verdadero cuando el planeador confirma que existe una ruta posible hacia el destino deseado, de lo contrario, toma el valor de Falso.***	***Planeador***	***Bool***

Tabla 4.10 Variables de estado de la arquitectura 3+.

El diagrama de estados de la figura 4.17 muestra los estados en los que puede encontrar el robot de acuerdo a los valores que presenten las diferentes variables que integran el modelo del mundo. El estado inicial propone el estado *Esperando*. Si la variable EXISTE_META es activada significa que un usuario ha indicado al robot que se mueva hacia algún lado. Cuando el coordinador detecta esta variable activa, pasa al estado Planeando, que significa hacer un llamado al componente de planeación y verificar que exista una ruta valida para mover el robot desde la ubicación actual al punto solicitado. Si la ruta existe, entonces el navegador moverá al robot a través de la ruta generada por el planeador.

En el proceso de navegación pueden suceder dos tipos de anormalidades:

La primera puede suceder cuando el robot no puede alcanzar la meta siguiendo la ruta generada debido al carácter dinámico del entorno (por ejemplo, si una puerta estuviera cerrada). Si esto sucede, se replantea una ruta alterna desde el punto actual hasta la posición meta. Este replanteamiento de ruta se presenta sólo hasta para 3 ocasiones, después de las cuales, la meta se considera inalcanzable y se pasa al estado inicial (*Esperando*).

La otra anormalidad sucede cuando el robot se *pierde* o no se puede localizar en el entorno actual. Este problema puede ser originado por diferentes causas, por ejemplo el deslizamiento de las ruedas sobre superficies resbaladizas causa problemas con la odometría del robot así como también se puede presentar el caso del *robot secuestrado* mismo que se caracteriza cuando se levanta y se coloca el robot en alguna posición que no corresponde a la ruta original. En este caso, se activa la variable PERDIDO y el robot pasa del estado *Navegando* al estado *Localizando*. Cuando el proceso de localización termina y el robot vuelve a estar dentro de una posición conocida, se replantea la ruta del punto actual al punto meta. Si el plan se ejecuta sin problema, el robot llegará a la posición meta. Cuando esto suceda, se activará la variable ESTA_EN_META, que indicará al coordinador que la siguiente acción a realizar es *Esperar* hasta recibir una nueva petición [Pacheco Sánchez José A. 2005].
.

Un punto de convergencia en el modelo de estado, es la variable BATERÍA _ BAJA. Cuando esta variable es activada, exceptuando el estado PERDIDO, no importa el estado en que se encuentre el robot ya que en ese momento realizará la acción de *Ir a base*, lo que significa que el robot conocerá *a priori* un sitio ubicado dentro del mapa de navegación (capturado por el usuario) para que reciba el mantenimiento pertinente. En el caso de PERDIDO, primero tendrá que localizarse para después trazar la ruta hacia la base.

En la figura 4.18, se muestra el diseño de la arquitectura 3+ modificada con dos coordinadores tolerantes a fallos.

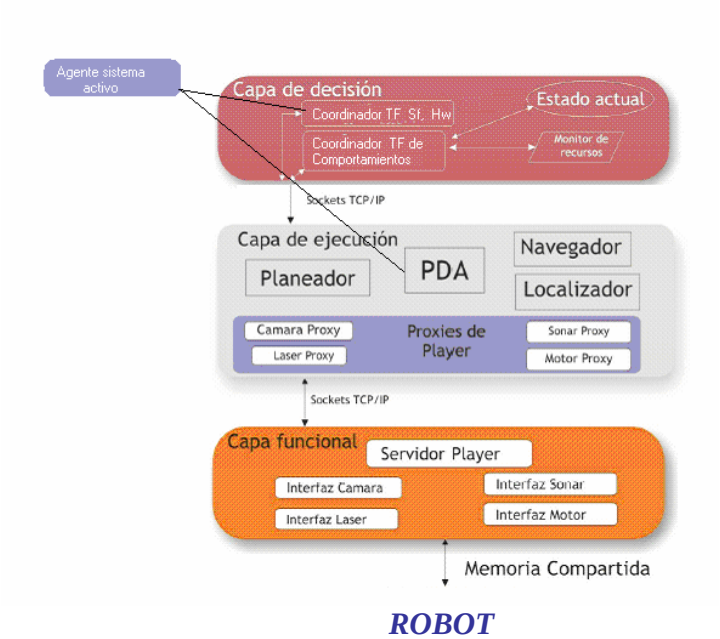


Figura 4.18 Arquitectura 3+ modificada con sus dos coordinadores tolerantes a fallos.

4.11 Conclusiones del capítulo

En este capítulo se realiza la modificación al diseño de una arquitectura física distribuida para el control de un robot móvil, de tal manera que se pueda implementar eficientemente la arquitectura tolerante a fallos propuesta en este trabajo, y así obtener una mayor operatividad en el robot.
La arquitectura seleccionada contempla reutilizar en forma de componentes las aplicaciones elaboradas en el contexto de desarrollo de aplicaciones y se lleva hacia servidores remotos aquellas que requieren de características de cómputo intensivo, como el procesamiento de video e imágenes. En la arquitectura 3+, como las características del modelo extendido y la integración de los módulos de navegación, localización, planeación e interfaz con un dispositivo móvil tipo PDA. Esta arquitectura por sus características especiales se seleccionó implementar la capa de tolerancia a fallos mediante un Sistema Multiagente que es el objetivo principal de este trabajo. Por lo que se tuvo que modificar la capa de decisión analizo implementando un coordinador tolerantes a fallos que sirve como interfase al Sistema Multiagente diseñado en este trabajo que tolera los fallos a nivel software y hardware y se terminó de diseñar el coordinador que ya estaba implementado en esta arquitectura para tolerar fallos a nivel comportamiento, implementando mensajes y reconfiguración en caso de ocurrir dichos fallos.

Por tal motivo la capa de decisión con sus coordinadores nos dio la pauta para implementar uno nuevo que sirviera de internase entre la arquitectura 3+ y nuestro Sistema que tolera los fallos a través de una red de agentes, esto se logra de una manera relativamente sencilla sin necesidad de realizar cambios drásticos en la arquitectura software 3+.

Fue necesario realizar las especificaciones del sistema detalladamente ya que son la base principal para poder modelas nuestro SMA con la Metodología MaSE y poder hacer llevar a cabo una metodología de desarrollo tolerante a fallos como se describe en el capítulo 3.

Hacer las especificaciones de nuestro sistema fue la clave para realizar adecuadamente el modelo de nuestro SMA, y no fue una tarea fácil, ya que se requiere además de conocer muy bien la funcionalidad y operabilidad del sistema a modelar, paciencia, organización y un método que cada diseñador desarrolla para no dejar de lado ninguna especificación, ya que si no están todos los detalles, hasta el más mínimo que sea en esta fase, no podemos obtener eficientemente un SMA a nuestra medida.

Capítulo 5 Análisis de fiabilidad

5.1 Introducción

Un robot móvil es un sistema complejo integrado por mecanismos móviles, un subsistema sensorial de visión, un subsistema de manipulación, y varias clases de sensores de inspección, etc. Por lo tanto el controlador del robot contiene gran cantidad de componentes de hardware. Es está de sobra decir, que cada componente en el sistema del robot debe tener suficiente fiabilidad. Por lo que la fiabilidad de su sistema de control es particularmente importante, ya que trata con los dispositivos críticos e importantes que manejan directamente a los actuadores.

Por tal motivo la fiabilidad y la tolerancia a fallos están siendo de gran importancia. La necesidad de producir robots altamente confiables ha creado el interés en los científicos por estudiar las diferentes herramientas que se utilizan en el diseño de mecanismos tolerantes a fallos. Tales herramientas se basan en la ingeniería de la fiabilidad que intentan evaluar la eficacia o efectividad de estos nuevos diseños, y así mismo ayuda a desarrollar algoritmos para detectar y aislar los fallos en los componentes que integran al robot usando relaciones analíticas de la redundancia [Wen ChunYun Cai KaiYuan, Zhang Ming Lian 1991].

Los componentes redundantes que se requieren para tolerar los fallos en el diseño de un robot aumentan los costos y la posibilidad de tener fallos. Las herramientas de análisis de fiabilidad tales como árboles de fallos y modelos de Markov son indispensables para obtener información confiable que demuestren las ventajas de diseñar sistemas robóticos que cuentan con mecanismos tolerantes a fallos [Joanne Bechta Dugan, Stacy A. Doyle 1996].

Desafortunadamente, los cálculos que se realizan para obtener los porcentajes de fallos en los componentes que integran un sistema robótico frecuentemente son dependientes de la configuración del robot y el ambiente donde éste trabaja, la mayoría de las veces se realizan aproximaciones en dos aspectos durante la fase de su diseño. Donde se aplica un único valor a los porcentajes de fallos, y es probable que arroje un resultado incorrecto. Una mejor manera es considerar el rango completo de los porcentajes. Ya sea que se aplique un solo valor o toda la distribución a las características del fallo es muy probable que de un resultado falso.

El paradigma de probabilidad es muy utilizado en el análisis de fiabilidad, pero estudios realizados recientemente han aportado información para creer que no es el adecuado cuando se cuenta con pocos datos, por lo que realizar un análisis basado solamente en probabilidad puede ser cuestionable para los sistemas con requisitos muy específicos, tal es el caso de un robot.

Por otra parte la ingeniería de fiabilidad esta normalmente soportada por un modelo de probabilidad, que a menudo es inapropiado para realizar este tipo de tarea en los sistemas robóticos ya que frecuentemente se carece de información probabilística durante la fase del diseño, y además los cálculos probabilísticos son complejos y no intuitivos, dando por resultado un análisis difícil de entender [Boris Gnedenko, Igor Ushakov 1995]

Los análisis realizados bajo el paradigma puramente probabilístico requieren generalmente gran cantidad de información sobre el sistema que se requiere analizar, para poder obtener confiablemente

los porcentajes de fallos, fallos acumulativos, o índices de vida de los componentes que integran al sistema.

Actualmente la lógica difusa ofrece una alternativa al paradigma de probabilidad, cediendo el paso al paradigma de *posibilidad* que se considera más apropiado para realizar análisis de fiabilidad en el contexto de la robótica [Leuschen Martin 1997]. Las matemáticas aplicadas al *paradigma de la posibilidad* permiten realizar cálculos cuantitativos de fiabilidad que preservan la incertidumbre presente de los datos originales. El modelo basado en *posibilidad* se ocupa de la incertidumbre de una manera que evita realizar asunciones injustificables, realizando solamente las asunciones que son requeridas de una manera clara a través del análisis. Las muestras pequeñas son manejadas fácilmente por éste paradigma, haciéndolo de una manera satisfactoria y modificando los requisitos particulares, que son aspectos importantes de la fiabilidad en robótica.

La lógica difusa mejora el análisis de fiabilidad logrando así confiabilidad con el concepto de utilidad. Los modelos confiables estándares asumen generalmente una representación binaria del fallo. El sistema está trabajando o está en fallo. Un modelo más efectivo permite que los sistemas se degraden lentamente cuando fallan instantáneamente. La utilidad permite fácilmente la representación de los fallos. Por estas razones, existe una motivación considerable para adaptar los métodos basados en la probabilidad tradicional de la fiabilidad, al contexto de la lógica difusa.

Una de las herramientas más comunes en el área de fiabilidad, son los árboles de fallos y sus variantes que se les ha implementado lógica difusa en diferentes grados. A pesar que los árboles de fallos son muy útiles, se limitan algo en sus aplicaciones. Sin embargo, se han tenido grandes progresos últimamente en el análisis de fiabilidad mediante los árboles de fallos, por lo que se ha diseñado una gran variedad de software profesional para ser aplicado en la industria.

Este trabajo se utiliza las herramientas de árboles de fallos en un contexto difuso posibilistico para realizar un análisis de fiabilidad en los componentes que integran el sistema de control distribuido de un robot móvil.

En la sección 5.2 se realiza estudio de los árboles de fallos para realizar análisis de fiabilidad. En la sección 5.3 Se investiga la teoría de los conjuntos difuso En la sección 5.4 se realiza el estudio de árboles de fallos difusos. La sección 5.5. La sección 5.6 se da las conclusiones del capítulo

5.2 El árbol de fallos herramienta estándar para realizar análisis de fiabilidad

Investigaciones previas se han enfocado a proveer tolerancia a fallos a través de duplicar los componentes que integran al sistema, tales como; los motores o sensores para el caso de hardware. Aunque la redundancia es una herramienta útil para la tolerancia de fallos, el duplicar partes incrementa el tamaño del robot, el costo en su construcción, el peso, y la inercia que afectan al controlador del robot.

Muchos de los robots estáticos avanzados actualmente no cuentan con motores redundantes para ayudar a tolerar fallos en este tipo de componente, estos robots poseen grados de libertad redundantes que permiten configuraciones múltiples del robot para una misma posición del actuador. Esta redundancia cinemática se utiliza principalmente para proporcionar más opciones con la finalidad de evitar obstáculos, por lo que los grados de libertad adicionales permiten al robot soportar varios fallos sin perder su gama en el movimiento. Para tolerar la pérdida de un componente en un robot móvil por ejemplo; el fallo en uno de los sensores infrarrojo instalados alrededor de un anillo, la información

sobre el fallo es detectado y se retransmite al planificador que ajusta el plan para redistribuir la carga de trabajo del sensor que falló entre los demás sensores libres de fallo. La mayoría de las veces los diseñadores de robots al implementarle tolerancia a fallos, tienden ha utilizar solamente esquemas que confían en la redundancia física de componentes que lo integran (hardware). Sin embargo existen muchos métodos para tolerar fallos que no necesariamente alteran al sistema físico.

5.2.1 Árboles de fallos

El Análisis de Árbol de Fallos (AAF) es un método probabilístico, cuantitativo y deductivo, que se centra en un fallo proporcionando un método para determinar las causas que lo han producido. Fue introducido en la década de los 60's para verificar la fiabilidad en el diseño del cohete Minuterman y ha sido ampliamente utilizado desde entonces, se ha experimentado un considerable esfuerzo de desarrollo, tanto teórico como práctico. Las aplicaciones de la técnica se han extendido hacia múltiples campos de la tecnología y de otras disciplinas como la Economía, Medicina, Centrales Nucleares, Robótica, Plantas Químicas, etc. Su utilización se basa en que proporciona resultados cualitativos mediante la búsqueda de rutas críticas, así como resultados cuantitativos en términos de probabilidades de fallos de los componentes [Muñoz Martinez V. F. 2004].

Para el tratamiento del problema se utiliza un modelo gráfico que muestra las diferentes combinaciones de los fallos en los componentes. La técnica consiste en un proceso deductivo basado en las leyes de álgebra de Boole, que permite determinar declarar sucesos complejos estudiados en función de los fallos básicos de los elementos que interviene.

Una desventaja es que no existe una manera de asegurar que todas las causas que han provocado el fallo se han evaluado. El diseñador tiende a identificar los eventos más importantes o los más obvios que podrían causar un determinado fallo. Sin embargo, los eventos que no son modelados normalmente tienen una probabilidad baja de ocurrir y se puede hacer caso omiso de estos o tratarse como un solo evento básico sin excesivamente predisponer el análisis [Walker Ian D., Visinsky M. L.,Cavallaro. J. R 1991].

La principal desventaja de este enfoque radica en la dificultad de la construcción del árbol por sí mismo. Los trabajos de investigación en esta área se han concentrado en el desarrollo de métodos sistemáticos para la construcción de los árboles de fallos utilizando modelos del comportamiento de los componentes e información topográfica sobre la interconexión entre los componentes.

Son cuatro las etapas que se pueden distinguir en un Análisis con Árboles de Fallos (AAF):

- Definición del sistema.
- Construcción del árbol de fallos.
- Análisis cuantitativo.
- Análisis cualitativo.

En la *Definición del Sistema* se debe recolectar información y especificaciones de todo tipo sobre el sistema que se esta estudiando, como pueden ser: Diagramas de flujo o de bloques, definiciones de sus objetivos, modos de funcionamiento, procedimientos, mapas de relación de datos y posibles fallos, etc. Todo esto es con la finalidad de diseñar e implementar una base de datos lo más completa posible del sistema que se esta analizando [Vasey W.E., Goldberg, F. F., Roberts, N. H., Haasl D.F. 1981].
.

Construir el árbol de fallos es una tarea difícil cuando los sistemas son demasiado grandes o complejos. Actualmente existe software que ayuda realizar éste trabajo, entornos sobre Windows, así como entornos que se desarrollan sobre Sistemas Expertos [John D. Healy 1996.].

Análisis cualitativo en esta etapa se debe de obtener información importante del sistema bajo estudio, tomada de la estructura del árbol de fallos ya construido. Aunque es muy variada la información a obtener [Walter Ian D., Visinsky M. L.,Cavallaro. J. R 1991], los Grupos de Corte Mínimos (GCM), que son las combinaciones mínimas de eventos básicos (entradas del árbol de fallos), que conducen a la aparición del suceso no deseado bajo estudio (suceso superior o *top* del árbol). Importantes conclusiones se pueden realizar con el estudio de la composición de estos GCM, por lo que se considera una etapa muy importante.

Análisis cuantitativo consiste en obtener medidas relacionadas con el evento no deseado (*top* del árbol). Son numerosas las medidas que se pueden obtener en un árbol de fallos, entre estas se pueden mencionar la frecuencia esperada de ocurrencias, la importancia de cada elemento del árbol (y por tanto del sistema) y la probabilidad del suceso *top*.

A lo largo de los años se han desarrollado diferentes metodologías para la evaluación cuantitativa; en la década de los 60's hasta mediados de los 70´s, fueron muy aceptados los procedimientos de simulación por el método de Monte Carlo, en los que se definieron métodos muy eficaces, como fue el caso del denominado Muestreo Daga .

AL final de los 70's y principios de los 80's, se desarrollaron potentes algoritmos de evaluación basados en recorrer las ecuaciones booleanas del árbol tanto ascendentemente (desde las causas básicas hasta el evento *top*), como descendentemente (desde el suceso *top* hacia las causas básicas). Estos algoritmos constituyeron un gran avance y causaron el abandono casi completo de los métodos basados en simulación [John D. Healy 1996.].

Al final de la década de los 80's, los avances en métodos de programación, propiciaron el desarrollo de nuevos métodos, denominados directos, que evaluaban los árboles de fallos mediante programas que explotaban características específicas de los nuevos paradigmas de los lenguajes, tal fue el caso de la Programación Orientada a Objetos, o el uso del Teorema de Factorización.

La complejidad de los sistemas técnicos de la década de los 90's, marcada entre otras características por la inclusión de los microprocesadores en los sistemas, aumentó considerablemente las exigencias en cuanto a potencia de los métodos de evaluación cuantitativa. La década inicia con una gran aportación que consistía en la conversión de los árboles de fallos a estructuras equivalentes denominadas Diagramas de Decisión Binarios (DDB), con ello se consiguió evaluar en tiempos de CPU aceptables árboles de un tamaño nunca antes abordado. A mediado de la década de los 90´s se ha incrementado considerablemente los trabajos de desarrollo de métodos de diseño basados en computadora, para los cuales son necesarios métodos de evaluación muy rápidos pero también confiables, ya que es necesario evaluar miles o millones de árboles de fallos diferentes como parte de un proceso de optimización. Esto ha sido posible, entre otras aportaciones, por la introducción de la posibilidad de contemplar alternativas de diseño en un mismo árbol analizando [Vasey W.E., Goldberg, F. F., Roberts, N. H., Haasl, D.F. 1981].

5.2.1.1 Descripción del método de análisis

Se trata de un método deductivo de análisis que parte de la previa selección de un suceso no deseado o evento que se pretende evitar, sea éste un accidente de gran magnitud (explosión, fuga, derrame, etc.) o

sea un suceso de menor importancia (fallo de un sistema de cierre, etc.) para averiguar en ambos casos los orígenes de los mismos.

Para ser eficaz, un análisis por árbol de fallos debe ser elaborado por personas profundamente conocedoras de la instalación o proceso a analizar y que a su vez conozcan el método y tengan experiencia en su aplicación; por lo que, si se precisa, se deberán constituir equipos de trabajo pluridisciplinarios (técnico de seguridad, ingeniero del proyecto, ingeniero de proceso, etc.) para proceder a la reflexión conjunta que el método propicia. Consiste en descomponer sistemáticamente un suceso complejo denominado *suceso TOP* en *sucesos intermedios* hasta llegar a *sucesos básico*, El suceso TOP ocupa la parte superior de la estructura lógica que representa el árbol de fallos. Es el suceso complejo que se representa mediante un rectángulo.

Los sucesos intermedios: son encontrados en el proceso de descomposición y que a su vez pueden ser de nuevo descompuestos. Se representan en el árbol de fallos como rectángulos. Los sucesos básicos: Son los sucesos terminales de la descomposición. Pueden representar cualquier tipo de suceso: sucesos de «fallos», error humano o sucesos de «éxito»: ocurrencia de un evento determinado. Se representan en círculos en la estructura del árbol analizando [Vasey W.E., Goldberg, F. F., Roberts, N. H., Haasl, D.F. 1981].

Los sucesos no desarrollados: Existen sucesos en el proceso de descomposición del árbol de fallos cuyo proceso no prosigue, ya sea por falta de información, o porque no se considera necesario. Se representan mediante un rombo y se tratan como sucesos básicos (ver tabla 5.1).

El árbol de análisis de fallos produce:

1.- Representación gráfica de la cadena de eventos y condiciones que llevan al fallo de un componente.
2.- Identificación de aquellos contribuidores potenciales que son críticos.
3.- Comprensión mejorada de las características del sistema.
4.- Introducción a la probabilidad cualitativa y cuantitativa de la pérdida seleccionada para el análisis.
5.- Identificación de los recursos encargados de prevenir el fallo.
6.- Guía de despliegue de recursos para optimizar el control de un riesgo.
7.- Documentación de algún resultado critico

Símbolo	Significado del símbolo
	SUCESO BÁSICO norequiere de posterior desarrollo al considerarse un suceso de fallo básico.
	SUCESO DESARROLLADO no puede ser considerado como básico, sus causas no se desarrollan, ya sea por poco interés o falta de información.
	El susceso de salida ´s´ ocurrirá si y solo si ocurren todos los sucesos de entrada E1 , B1
	El suceso de salida ´s´ si ocurre uno o más de los sucesos de entrada E1,B1.
	SÍMBOLO DE TRANSFERENCIA indica que el árbol continúa en otro lugar.
	PUERTA DE INHIBICIÓN la salida ocurrirá si y solo si lo hace su entrada y además y ademas se satisface una condición dada X.
	SUCESO INTERMEDIO resultante de una combinación de sucesos más elementales por medio de puertas lógicas , así mismo representa el suceso deseado del que parte el árbol.
	PUERTA ´Y´ PRIORITARIA el suceso de salida ocurrirá si y solo si todas las entradas ocurren en una secuencia determinada, que normalmente se especifica con una elipse a la derecha.
	PUERTA ´O´EXCLUSIVA el suceso desalida ocurrirá si lo hace una de las entradas, pero no dos o más de ellas.

Tabla 5.1 Símbolos utilizados para la representación del árbol de fallos.

En el proceso de descomposición del árbol se recurre a una serie de puertas lógicas que representan los operadores del álgebra de sucesos. Los dos tipos más elementales corresponden a las puertas AND y OR cuyos símbolos se indican a continuación analizando [Vasey W.E., Goldberg, F. F., Roberts, N. H., Haasl, D.F. 1981]:

La puerta OR se utiliza para indicar un «0» lógico: significa que la salida lógica S ocurrirá siempre y cuando ocurran por lo menos una de las dos entradas lógicas e1 o e2 (figura 5.1).

La puerta AND se utiliza para indicar un «Y» lógico. Para que ocurra la salida lógica S es necesario que ocurran conjuntamente las dos entradas lógicas e1 y e2 (figura 5.1).

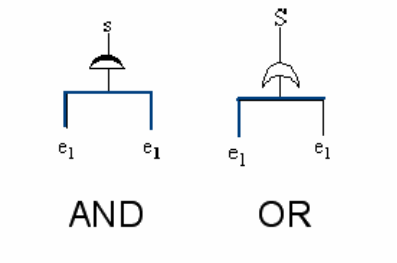


Figura 5.1 Puertas AND y OR que constituyen el árbol de fallos.

5.2.1.2 Elaboración del árbol de fallos

En esta fase se integran todos los conocimientos sobre el funcionamiento y operación de la instalación con respecto del suceso estudiado.

El primer paso consiste en identificar el suceso *no deseado* o suceso TOP que ocupará la cúspide de la estructura gráfica representativa del árbol. De la definición clara y precisa del TOP depende todo el desarrollo del árbol. Con el suceso TOP se establecen de forma sistemática todas las causas inmediatas que contribuyen a su ocurrencia definiendo así los sucesos intermedios unidos a través de las puertas lógicas.

El segundo paso consiste en identificar a los contribuidores del primer nivel. Tercer paso ligar a los contribuidores al evento TOP mediante las compuertas lógicas. El cuarto paso consiste en identificar a los contribuidores del segundo nivel. Quinto paso ligar a los contribuidores del segundo nivel al evento TOP mediante las compuertas lógicas. Sexto pasos repetir hasta llegar a los eventos básicos.

La identificación del evento TOP requiere de: Explorar los registros históricos del componente, identificar contribuidores potenciales de fallos, desarrollar escenarios preguntándose que pasa si. Los eventos TOP, casi siempre representa pérdida potencial (alto riesgo). El evento TOP debe ser un fallo independiente o la condición del fallo (descrita típicamente por un sustantivo, un verbo, y especificando modificantes)

El efecto es el evento TOP y la causa cada evento contribuyente, los niveles abajo del evento TOP tiene que estar relacionado con una compuerta lógica y debe de contribuir de inmediato con el nivel superior.

En la figura 5.2 se muestra el árbol el fallo de los posibles eventos requeridos por lo que un motor no puede arrancar, el fallo se coloca en la cima o tope del árbol. Dicho fallo se puede deber a un problema en la bomba de gasolina, que a su vez puede tener el fallo por un filtro bloqueado o por una válvula obstruida. La rama derecha indica que el motor no arranca por problemas de encendido. El diseño del árbol de fallos implica desarrollar para cada fallo en la cima toda la sucesión de posibles problemas que pudieran afectarle.

El sistema que se presenta anteriormente es muy simple. El análisis de árboles de fallos para grandes sistemas como centrales nucleares, aviones, control de vuelos en aeropuertos o robots móviles, es mucho más complejos y no se puede resolverse manualmente. En la práctica se utiliza software especial que ayudan al diseño de los árboles de fallo y su análisis. El análisis pasa por la reducción del árbol y para ello se utilizan técnicas matemáticas basadas en la definición de subconjuntos de los eventos elementales que describen de manera razonablemente en el árbol analizando [Vasey W.E., Goldberg, F. F.,Roberts, N. H., Haasl, D.F. 1981].

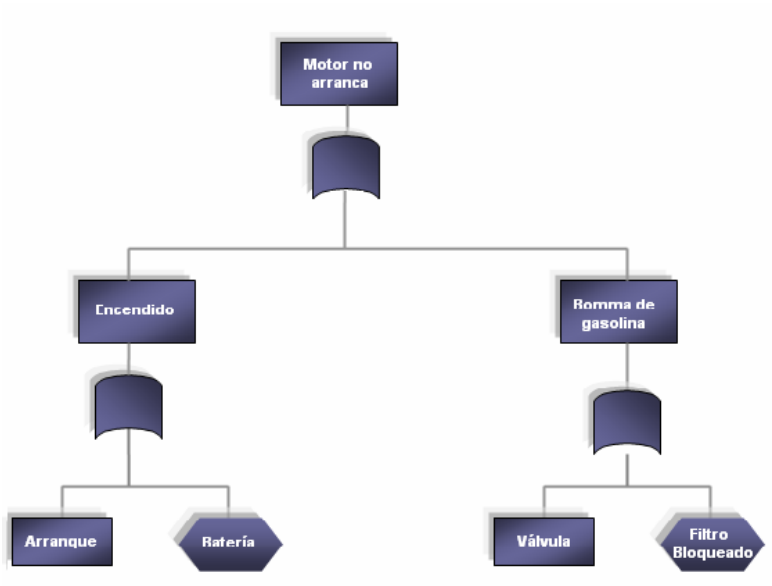


Figura 5.2 árbol de fallos que representa los fallos por lo que un motor no arranca.

5.2.1.3 Exploración del árbol

La exploración del árbol de fallos puede limitarse a un tratamiento *cualitativo* o acceder a un segundo nivel de análisis a través de la *cuantificación* cuando existen fuentes de datos relativas a las tasas de fallo de los distintos componentes [John D. Healy 1996.].

5.2.1.4 Evaluación cualitativa

Consiste en analizar el árbol sobre el plano de su estructura lógica para poder determinar las combinaciones mínimas de sucesos básicos que hagan que se produzca el suceso no deseado o evento que se pretende evitar (noción de conjunto mínimo de fallos) [Sutton Ian 2004].

Además, la estructura lógica de un árbol de fallos permite utilizar el álgebra de Boole, traduciendo esta estructura a ecuaciones lógicas. Para ello se expone muy brevemente tal sistema de equivalencia lógica:

1. Una puerta **OR** equivale a un signo **(+)**, no de adición sino de unión en teoría de conjuntos.
2. Una puerta **AND** equivale a un signo (.) equivalente a la intersección.

De esto se extraen las siguientes consecuencias:

- Transformar el árbol de fallos en una función lógica.
- La posibilidad de simplificar la función lógica del árbol debido a la constatación de falsas redundancias. La reducción de falsas redundancias (reducción booleana) consiste en simplificar ciertas expresiones booleanas y consecuentemente los elementos de estructura que las mismas representan.

En lo anterior se debe de notar la importancia de identificar durante el análisis, además de los fallos individuales de los componentes, los posibles fallos debidos a una causa común o la determinación de los componentes que fallan del mismo modo.

Para la resolución de árboles de fallos se realizan los siguientes pasos:

1. Identificación de todas las puertas lógicas y sucesos básicos.
2. Resolución de todas las puertas en sus sucesos básicos.
3. Eliminación de los sucesos repetidos en los conjuntos de fallo: aplicación de la propiedad idempotente del álgebra de Boole.
4. Eliminación de los conjuntos de fallo que contengan a su vez conjuntos de fallo más pequeños, es decir, determinación de entre todas las combinaciones posibles, los conjuntos mínimos de fallo: aplicación de la ley de absorción del álgebra de Boole.

En el caso de árboles sencillos (figura 5.2), los conjuntos mínimos de fallos se pueden obtener sustituyendo las puertas OR (Y) por sus entradas en las columnas de una matriz y las de las puertas AND(O) en las filas (figura 5.3) [Shaley D.M., Tiran J 2007].

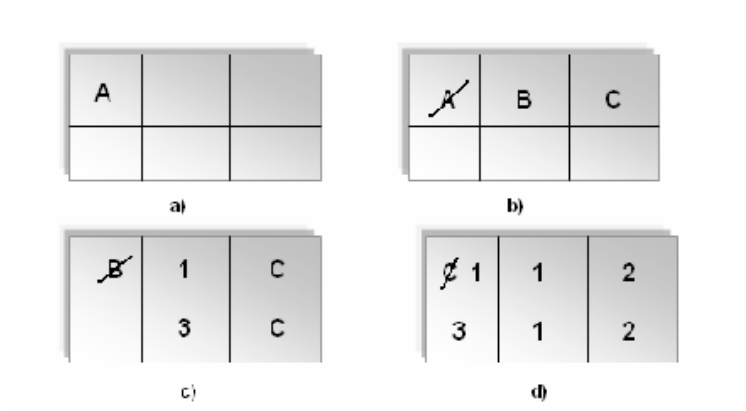


Figura 5.3 obtención del conjunto mínimo de fallos.

Se trata de ir descendiendo en el árbol para su resolución eliminando y sustituyendo los sucesivos símbolos de identificación de las puertas hasta obtener las diferentes combinaciones de fallos primarios identificados.

De la resolución del árbol de fallos se obtiene:

- Vía secuencial de fallos básicos generadores del acontecimiento final: 1.2 y 1.2.3.
- Conjunto mínimo de fallos que son necesarios para que se produzca el acontecimiento final: 1.2.

La vía 1.2.3 en realidad es la misma que la 1.2, ya que el evento sucede con la simultaneidad de los fallos 1 y 2 sin necesidad de que acontezca el fallo 3, con lo que el conjunto mínimo de fallos es 1.2.

5.2.1.5 Evaluación cuantitativa

Precisa conocer la indisponibilidad o probabilidad de fallo de aquellos sucesos que en el árbol se representan en un círculo (sucesos básicos) y determinar valores probabilísticos de fallo a aquellos sucesos que se representan en un rombo (sucesos no desarrollados).

Según el modo de fallo del componente, se calcula la probabilidad de fallo del mismo en función de la tasa de fallo que se puede obtener de base de datos y, fundamentalmente, de la propia experiencia. Existe, asimismo, información que proporciona datos estimativos sobre tasas de errores humanos que permite asignar valores probabilísticos a la ocurrencia [Jung W.S., Yang J.E., Ha J. 2006]. El conocimiento de los valores de probabilidad de los sucesos primarios (básicos o no desarrollados) permite:

- Determinar la probabilidad global de aparición del *suceso no deseado* o *evento que se pretende evitar*.
- Determinar las vías de fallo más críticas, es decir, las más probables entre las combinaciones de sucesos susceptibles de ocasionar el *suceso no deseado*.

Para la valoración de la probabilidad global de aparición del *suceso no deseado* se realizan los siguientes pasos:

1. Se asignan valores probabilísticos a los sucesos primarios.
2. Se determinan las combinaciones mínimas de sucesos primarios cuya ocurrencia simultánea garantiza la aparición del *suceso no deseado*: establecimiento de los *conjuntos mínimos de fallos*.
3. Se calcula la probabilidad de cada una de las vías de fallo representada por los conjuntos mínimos de fallos, la cual es igual al producto (intersección lógica en álgebra de Boole) de las probabilidades de los sucesos primarios que la componen.
4. Se calcula la *probabilidad de que se produzca el acontecimiento final*, como la suma de las probabilidades (unión lógica de todos los N conjuntos mínimos de fallo en álgebra de Boole) de los conjuntos mínimos de fallo, como límite superior, ya que matemáticamente debería restarse la intersección de éstos.

Ejemplo, el caso del árbol representado en la figura 5.1 se asignan valores medios de probabilidades de fallo a los sucesos primarios:

1. $P_1 = 5 \cdot 10^{-3}$; $P_2 = 6 \cdot 10^{-2}$; $P_3 = 10^{-3}$.
2. Conjunto mínimo de fallos: P_1 y P_2.
3. $P_{\text{vía (1)}} = P_1 \cdot P_2 = 5 \cdot 10^{-3}$ x $6 \cdot 10^{-2} = 300 \cdot 10^{-6}$.
4. Probabilidad de acontecimiento final: $P_{AF} = P_1 \cdot P_2 = 300 \cdot 10^{-6}$.

En este caso coincide con la probabilidad del conjunto mínimo de fallos ya que éste es único. En el supuesto que se plantea a continuación, en que el árbol que se desarrolla es ligeramente más complejo, se observará cómo se calcula la P_{AF} a partir de la existencia de varios conjuntos mínimos de fallos.

5.2.1.6 Análisis de fiabilidad con árboles de fallos en sistemas robóticos

Para desarrollar los algoritmos de detección de fallos y determinar la interdependencia de los fallos en el sistema de control de un robot es necesario basarse en la estructura existente del robot, para poder detallar los fallos. Herramientas útiles para desarrollar estas tareas es el análisis; ya sea por medio de árboles de fallos normales o difusos, o cadenas de Markov normales o difusas, etc.

En los robots, existen una multitud de posibles eventos que producen fallos, ejemplo en el caso que se tengan cables pelados se puede dar interrupciones en el funcionamiento del robot, y se pueden romper algunos componentes. El robot por sí solo no puede distinguir entre dos diferentes tipos de fallos, ya que solamente se puede percatar del efecto de estos. Por ejemplo, si un motor no está dirigiendo una la

llanta y el controlador le ha ordenado que lo haga, el motor puede no dirigir dicha llanta ya sea porque éste es el que tiene el fallo, o bien porque la llanta tiene el fallo, el controlador no puede determinar por si mismo que es lo que realmente sucede, el controlador del robot solo se puede percatar de que el motor se ha detenido y no trabaja.

Por este motivo, los eventos de los fallos se pueden detallar realizando un análisis con los árboles de fallos, y la mayor parte de las veces se determina basándose en una inspección visual en la estructura del robot [Wu W., Rao S.S. 2006].

De tal manera que si se realiza un minucioso detalle del árbol de fallos este puede ser muy útil para que podamos entender y trazar las posibles interacciones estructurales de fallos en el robot, y acentuar las capacidades de tal manera que se puedan detectar los fallos, sin embargo, los árboles se deben modificar para centrarse en la interacción funcional del fallo. En la práctica, los eventos del árbol se deben agrupar dentro de eventos que puedan detectar fallos en los sensores, o que ayuden al diseñar algoritmos inteligentes para detección de fallos en cualquier componente del robot. Si se realiza un análisis completo del sistema de control en un robot nos revela que el buen estado de los motores y los sensores son una consideración importante en la supervivencia de un robot [Visinsky L Monica 2003].

Analizar probabilidades de fallos y su propagación a través de los árboles de fallos permite que el diseñador realice los mecanismos tolerantes a fallos centrándose en los componentes que son más probables de tener un fallo cuando el robot esta en funcionamiento, siendo estos los más importantes y así diseñar sus procedimientos de detección de fallos. Incluso se puede conocer el orden de las magnitudes relativas en los fallos ayudando al diseñador a enfrentar los eventos inverosímiles del fallo. Por ejemplo, si hay solamente una pequeña posibilidad de que una llanta se pueda ponchar durante una tarea típica del robot.

Una desventaja sugerida del análisis con árboles de fallo es que no existe una manera de asegurarse de que todas las causas de un fallo se han evaluado, ya que el diseñador tiende a identificar los eventos mas importantes o los mas obvios que causarían una fallo dado. Sin embargo, los eventos que no son modelados normalmente tienen una baja probabilidad de ocurrir y se puede hacer caso omiso de estos o tratarse como un solo evento básico sin excesivamente predisponer el análisis. Varios fallos se pueden interconectar creando ramas o ciclos laterales en el árbol de fallos. En algunos robots móviles que tienen brazos, un movimiento en una articulación se puede juntar con otro movimiento de tal manera que el fallo de cualquier movimiento causa un fallo del otro movimiento [Wu W., Rao S. S 2006].

Para mantener los árboles simples, los dos fallos se consideran uno con dos veces la probabilidad de ocurrir. Es también a veces difícil determinar la relación entre los fallos.

Algunos investigadores han desarrollado base de datos de árboles de fallos y por medio de un sistema experto pueden determinar las causas posibles de los fallos detectados, la base de datos genera una lista de causas posibles para la inspección y su la reparación. Sin embargo, los árboles que se producen cuando se realiza el análisis mediante árbol de fallos, muestran la interconexión estructural de fallos, sin revelar siempre la función que afectan los fallos. Realizar este tipo de análisis, se puede determinar las trayectorias significativas de los fallos dentro de los árboles de fallos y así poder detectar los fallos funcionales más críticos en el sistema del robótico.

El análisis de los árboles de fallos que se ha realizado en diferentes robots, como es el caso de MEDULA por Mónica Visinsky, ha revelado la importancia de centrarse en los fallos de los sensores o de los motores ayudando a desarrollar algoritmos que sirven para detectar estos rápidamente.

5.3 Lógica difusa y posibilidad

La teoría de lógica difusa es introducida por Lofti A. Zadeh (padre de la teoría de la posibilidad) es una herramienta matemática intuitiva y poderosa que se caracteriza por querer cuantificar esta incertidumbre.

¿Qué es la lógica Difusa? Una definición simple se puede decir que es una lógica alternativa a la lógica tradicional donde no únicamente los valores de respuesta a alguna pregunta que se realiza es "si o no", falso o verdadero, 0 o 1, sino que aquí se toman en cuenta cada uno de los valores existentes entre 0 y 1, por ejemplo 0.001, 0.02, 0.5, etc., para determinar en que proporción la respuesta que se obtiene es verdadera o falsa. Es decir que si el valor es 0.001 es más falso que verdadero y si obtenemos 0.9 es más verdadero que falso. A estos valores entre 0 y 1 se le llaman grados de pertenencia.
Además de los grados de pertenencia esta lógica también se basa en el uso de reglas del tipo IF…THEN….ELSE.

Actualmente esta técnica esta siendo muy utilizada en electrodomésticos como lavadoras, televisores, control de robots, en mecanismos con inteligencia artificial, etc. Todos ellos utilizando lógica difusa con el fin de hacer más preciso su funcionamiento [Cox E. 1992].

El propósito de esta sección es lograr un mejor entendimiento de esta tecnología, mostrando los beneficios que se obtiene al aplicarla en el análisis de fiabilidad para los sistemas robóticos.

Dentro de esta lógica se usan mucho los términos variables lingüísticas, universo de discurso, valor lingüístico, conjuntos difusos, grados de pertenencia, función de pertenencia nítida, función de pertenencia difusa.

Una *variable lingüística* como su nombre lo indica es una variable cuyos valores son palabras o frases de un lenguaje natural. Es aquella que vamos a calificar de forma difusa. Por ejemplo porcentaje de fallos.

Universo de discurso es el rango de valores que pueden tomar los elementos que poseen la propiedad expresada por la variable lingüística. En el caso de la variable lingüística porcentaje de fallos podría ser el conjunto de valores comprendidos entre 5% a 90%.

Valor lingüístico a las diferentes clasificaciones que efectuamos sobre la variable lingüística: en el caso del porcentaje de fallos, se podría dividir el universo de discurso en los diferentes valores lingüísticos: "bajo", "medio", "alto".Cada valor lingüístico tendrá un conjunto difuso asociado, de forma que nos referiremos a los conjuntos "bajo", "medio", "alto" asociados a la variable lingüística *porcentaje de fallos*.

Función de pertenencia es aquella aplicación que asocia a cada elemento de un conjunto difuso el grado con que pertenece al valor lingüístico asociado. Los conjuntos difusos son caracterizados por sus funciones de pertenencia la cual se representa con μ, por ejemplo: $\mu A(x)= 1$, y se lee de la siguiente manera*; el grado de pertenencia* μ al conjunto *A* del valor *x* es igual a 1 [Lee C.C. 1990].

Por lo que se puede decir que un *conjunto es difuso* cuando el concepto al que representa tiene una función de pertenencia difusa asociada a él.

Notación de conjuntos difusos:

Sea F un conjunto difuso definido sobre el universo U.

$$F = \{(u, \mu F(u))\} | u \in U$$

Indica que F está formada por todos los pares ordenados "u" y el resultado de la función de pertenencia para todo elemento u dentro del universo de discurso U.

5.3.1 Conjuntos difusos

Los conjuntos difusos tienen sus raíces en la Teoría de Conjuntos. Aunque los conjuntos que maneja esta lógica no son los ordinarios sino un nuevo tipo de conjuntos: los conjuntos difusos o Fuzzy, se trabaja con ellos de manera similar.

Si se tiene un conjunto A, y B es un subconjunto de A. En teoría convencional de conjuntos decimos que $B \subset A$ (B esta incluido en A).Y para indicar que un elemento x cualquiera pertenece al conjunto A su notación es: $x \in A$ [Kaehler Steven D. 2004].

Ahora, existe otra forma para expresar esta noción y consiste en el uso de una función de pertenencia. Es decir una función que mediante su valor nos indique, si *x* pertenece o no a (*A)* y en que medida

Así, si se llama $\mu A(x)$ a dicha función, se puede indicar:

$\mu A(x) = 1$ si $x \in A$
$\mu A(x) = 0$ si $x \notin A$

Aquí se está frente al caso en que la cuestión es si pertenece o no. Pero al explorar esto a la realidad y se puede analizar que no es tan sencillo en la vida real. Si se quisiera clasificar el clima de una ciudad con respecto a la temperatura en frío o calurosa, entonces se podría definir dos conjuntos (figura 5.4):

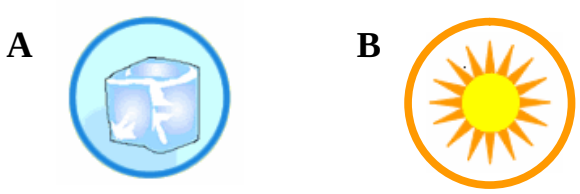


Figura 5.4 clasificación del clima en dos conjuntos frío y caluroso.

Por lo que se requiere definir la condición el clima para que sea *frío* tiene que tener una temperatura de *5ºC o menos.* Tomando a *x* como la *temperatura: μA(x)= 1,* SI x<= *5ºC* pertenece al conjunto frió, si *μA(x)= 0* Si x>*5ºC* pertenece al conjunto caliente. Este ejemplo basado en la lógica clásica es impreciso, si detectamos una temperatura de *5ºC* se puede decir que el clima está frío, pero si fuese 6ºC ya no está frío , porque no es igual ni menor que la temperatura marcada.

Sin embargo se siente frío cuando la temperatura es de 6ºC, por lo que el cambio entre conjuntos debe ser paulatino, se debe de cambiar de un conjunto a otro poco a poco. Debido esto nace la idea de usar funciones que indiquen pertenencia a conjuntos, cuyos resultados no sean exclusivamente unos y ceros, sino que puedan variar libremente en el intervalo de [1,0]. El conjunto difuso que representa lo frío es:

$$\mu(x) \begin{cases} 0 & :x>5°C \\ (x-1)/5 & :x\in[1°C, 5ºC] \\ 1 & :x <= 5°C \end{cases}$$

Si se aplica el método de conjuntos difusos. En el caso de la temperatura de 6ºC se puede decir que pertenece al conjunto de temperatura fría en un 0.99% es decir: $\mu A(x)= 0.99$ y además una pertenencia de: $\mu A(x)= 0.01$ al conjunto de temperatura calurosa. Estableciendo así que hace más frío que calor. Por lo qué formar parte de un conjunto no excluye la pertenencia a otro, en esta teoría [Kaehler Steven D. 2004].

Cuán es la finalidad de esta teoría, normalmente nuestra apreciación de la realidad no es tan simple como un Si o No. Por ejemplo un ciclista que sale a dar un paseo en su bicicleta se detiene de repente y dice que el clima esta muy caluroso, pero ¿Cuánto es caluroso?, ¿Exactamente 40ºC?, ¿o más?. No todos los seres humanos tenemos la misma sensibilidad hacia el clima, para una persona que vive en zonas desérticas donde las temperaturas llegan a subir cerca de los 50ºC, y va de vacaciones a un lugar como Acapulco donde la temperatura es 36ºC sentirán el clima agradable, en cambio para las personas que viven en lugares donde gran parte del año la temperatura oscila ente los 20ºC y 25ºC, en un clima desértico con temperatura de 48ºC sentirá demasiado calor. La lógica difusa tiene mucho que ver con la precisión.

La ventaja de la lógica difusa es que no todo es falso ni verdadero sino que se mueve en la escala del gris. Un sistema difuso ofrece una descripción más exible y más exacta de los adjetivos lingüísticos, en este caso particular del conjunto difuso que representa el frío, este conjunto no es posible representarlo mediante un sistema nítido, algunas temperaturas están más cerca del conjunto frío y menos cerca del conjunto caluroso, siendo el caso de los 6ºC, la región donde la temperatura se puede describir como fría con cierto grado de verdad se denomina difusa, los conjuntos difusos pueden interpretar el lenguaje humano y convertirlo a valores numéricos.

Con la lógica difusa se puede expresar la imprecisión matemáticamente de una manera sencilla y fácil de manipular, sin complicarnos la existencia con modelos matemáticos complejos, y se obtienen resultados certeros. La lógica difusa se propone como una herramienta para salvar estas ambigüedades, Obviamente no es la única manera, pero una de las más simple.

Los Conjuntos Difusos se pueden operar entre sí del mismo modo que los conjuntos clásicos. Puesto que los primeros son una generalización de los segundos, es posible definir las operaciones de intersección, unión y complemento haciendo uso de las mismas funciones de pertenencia:

Si un conjunto C es un conjunto que incluye los conjuntos A y B para cualquier x perteneciente a C se dice que:

A esta incluido en B si $\mu A(x) \leq \mu B(x)$ y se representa $A \subset B$.
A y B son iguales si $\mu A(x) = \mu B(x)$ y se representa $A = B$.
A y B son complementarios si $\mu B(x) = 1 - \mu A(x)$.
La intersección se indica: $\mu A \cap B(x) = \min(\mu A(x), \mu B(x))$
La unión se indica: $\mu A \cup B(x) = \max(\mu A(x), \mu B(x))$

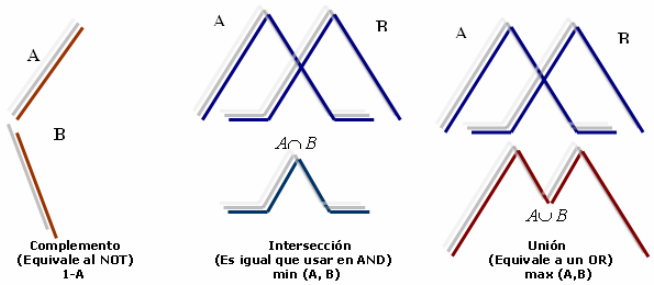


Figura 5.5 operaciones con conjuntos difusos

Resumiendo se puede tener que:

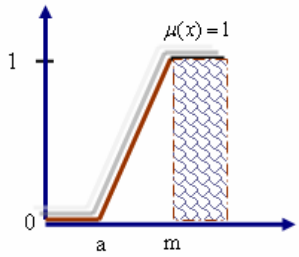
Unión $(\max(\mu_a(x),\mu_b(x))$
Intersección $(\min(\mu_a(x),\mu_b(x))$
Complemento $1- \mu_a(x)$

5.3.2 Funciones de pertenencia

Cualquier función puede ser válida para definir a los conjuntos difusos, se utilizará la que mejor represente al problema que se requiera resolver, según el comportamiento que se observen de los valores que se utilicen como entrada hacia los conjuntos difusos. Existen ciertas funciones típicas que se suelen usar, siendo estas:

Función GAMMA:

$$\mu(x)=\begin{cases}0 & \text{Si} \quad x \le a \\ (x-a)/(m-a) & \text{Si} \quad x \in (a,m) \\ 1 & \text{Si} \quad \boldsymbol{x \ge m}\end{cases}$$



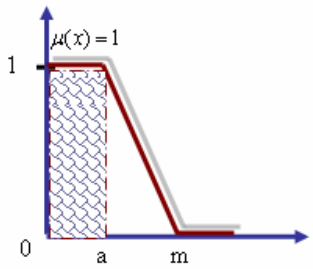
Es decir;

$$\mu(x)=\begin{cases}0 & \text{Para} \quad x \le a \\ \dfrac{(x-a)}{(m-a)} & \text{Para} \quad a < x < m \\ 1 & \text{Para} \quad x \ge m\end{cases}$$

Función L:

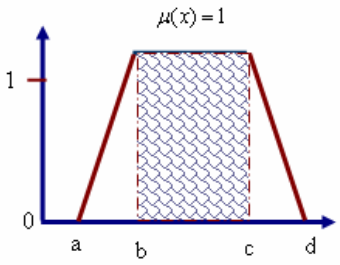
Puede definirse como 1 menos la la función GAMMA

Donde los valores de *x* en la función GAMMA tienen grado de pertenencia 1 aquí será 0 y viceversa.



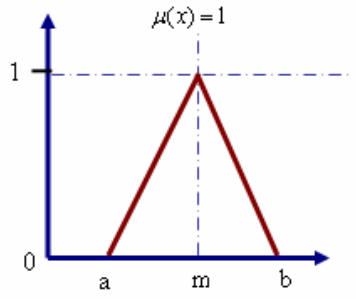
Función PI o trapezoidal:

$$\mu(x)=\begin{cases}0 & x \le a \\ \dfrac{(x-a)}{(b-a)} & a < x \le b \\ 1 & b < x \le c \\ \dfrac{(d-x)}{(b-c)} & c < x \le d \\ 0 & \boldsymbol{x > d}\end{cases}$$



Función LAMBDA o Triangular:

$$\mu(x) = \begin{cases} 0 & \text{Si} \quad x \le a \\ (x-a)/(m-a) & \text{Si} \quad x \in (a,m) \\ (b-x)/(b-m) & \text{Si} \quad x \in (m,b) \\ 0 & \text{Si} \quad x > b \end{cases}$$



Es decir;

$$\mu(x) = \begin{cases} 0 & \text{Para} \quad x \le a \\ \dfrac{(x-a)}{(m-a)} & \text{Para} \quad a < x \le m \\ \dfrac{(b-x)}{(b-m)} & \text{Para} \quad m < x \le b \\ 0 & \text{Para} \quad x > b \end{cases}$$

5.3.3Razonamiento difuso

La teoría de conjuntos difusos permite representar hechos y relaciones imprecisas.

> *Se entiende por **razonamiento difuso** el proceso de realizar deducciones a partir de hechos y relaciones difusas, así como la combinación de evidencias difusas y la actualización de la precisión de las creencias.*

Una proposición difusa simple es aquella que asigna un valor a una variable difusa, por ejemplo: La velocidad de ese automóvil es muy alta o La luz que emite ese foco es muy baja. Una proposición difusa tiene por tanto asociado un conjunto difuso ***A*** (el valor lingüístico asignado, muy alta en este caso) y su correspondiente función de pertenencia **μ** definida sobre el conjunto difuso A los elementos del universo de discurso *u* ∈ U.

Una *proposición difusa compuesta* es aquella que se obtiene mediante la combinación de dos o más proposiciones difusas simples, que pueden haber sido modificadas o no antes de la agrupación. Para agrupar proposiciones difusas simples es posible utilizar las conectivas **AND** y **OR,** y para modificar una proposición difusa simple se requiere utilizar el **NOT**. Así por ejemplo construir proposiciones difusas del tipo:

1.- "La distancia hacia aquella ciudad es corta" ***AND*** *"la velocidad del avión es muy alta"*
2.- "La distancia hacia aquella ciudad es larga" ***OR*** *"la velocidad del avión es baja"*
3.-"La distancia hacia aquella ciudad ***NOT*** *es alta"*

Los operadores lógicos difusos pueden definirse de forma análoga como se definieron las operaciones entre conjuntos: sean p y q dos proposiciones difusas, A y B los conjuntos difusos que intervienen en ellas, con funciones de pertenencia μA y μB definidas respectivamente sobre universos de discurso U y V.

Los operadores lógicos se utilizarán de la siguiente manera:

En la agrupación de proposiciones número "1" se utiliza el operador "AND" para unirlas, por lo tanto se genera una INTERSECCIÓN, es decir; para que se cumpla esta proposición, el valor *x* ***tiene que*** pertenecer tanto al conjunto difuso A como al B, y *x* tiene un grado de pertenencia con A ($\mu A(x)=?$) y otro con B ($\mu A(x)=?$).
Como es una intersección la que se esta ejecutando en esa proposición, el grado de pertenencia que se requiere tomar en cuenta para x ***es el menor de los dos.***

$$\mu A \cap B(x) = \min(\mu A(x), \mu B(x))$$

En la agrupación de proposiciones número "2" se utiliza el operador "OR" para unirlas, por lo tanto se genera una UNIÓN, es decir; para que se cumpla esta proposición, el valor *x* ***puede*** pertenecer al conjunto difuso A o al B o a ambos pero no es obligación que forme parte de los dos. En caso de que x se encuentre en ambos conjuntos tendrá un grado de pertenencia con A ($\mu A(x)=?$) y otro con B ($\mu A(x)=?$), como es una unión la que se esta ejecutando en esa proposición, el grado de pertenencia que vamos a tomar en cuenta para x ***es el mayor de los dos.***

$$\mu A \cup B(x) = \max(\mu A(x), \mu B(x))$$

En la proposiciones número "3" se utiliza el operador "NOT" por lo que se genera una NEGACIÓN, es decir; el valor *x* esta en el conjunto opuesto al mencionado en la proposición. Por ejemplo si el conjunto difuso A esta representada por el valor lingüístico "alta", el valor *x* puede tomar todos aquellos valores que no se encuentran en el conjunto A.

$$\mu B(x) = 1 - \mu A(x)$$

5.3.4 Reglas heurísticas

Una vez entendido el concepto de razonamiento difuso el siguiente paso es comenzar armar las reglas heurísticas o también llamadas de inferencia, las cuales son las que finalmente arrojan los resultados.

Se requiere definir lo que significa una implicación, es decir, se tiene que asignar una función de pertenencia a una agrupación *antecedente consecuente* del tipo $p \rightarrow q$. Definir el significado de la implicación permitirá razonar con reglas del tipo:

SI *"la temperatura es caliente"*
ENTONCES *"la velocidad de ventilación debe ser alta"*

Con la formalización anterior pueden representarse hechos y reglas difusas, y pueden realizarse inferencias aplicando reglas de inferencia. Se analizará el siguiente caso:

Inferencia difusa con antecedentes claros. Al supone que se tiene una regla difusa del tipo:

Si p ***ENTONCES*** q y un valor de entrada p*.

La conclusión será un hecho difuso q*, del cual se quiere conocer su función de pertenencia.

Ejemplo:

Teniendo la siguiente regla difusa:

p $\rightarrow$ q. = "**SI** la temperatura es *caliente*, **ENTONCES** la velocidad del ventilador debe ser *alta*".

Suponga que el valor de entrada con el que se cuenta fue recolectado mediante sensores instalados en ciertas áreas donde se detecto calor teniendo que:

p = "la temperatura de 40ºC"*

El hecho p* puede utilizarse para disparar la regla y obtener así un valor difuso para la *velocidad del ventilador* que debe aplicarse:

Dato de entrada p*: Temperatura = 40ºC
Regla Heurística: SI temperatura = caliente ENTONCES velocidad de ventilador = alta
Resultado = velocidad de ventilación = q*

Se supone que las funciones de pertenencia de los conjuntos difusos A = caliente y B = alta son los que se dan en la siguiente figura 5. 6:

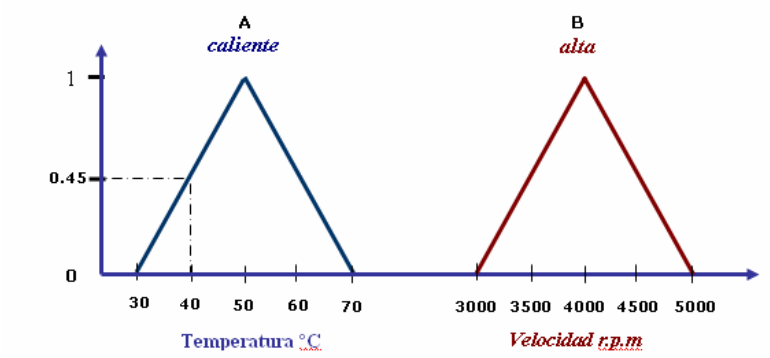


Figura 5.6 Conjuntos difusos representan la temperatura y la velocidad, se utiliza una función de pertenencia tipo triangular.

El al figura 5.6 se puede apreciar el valor de entrada p* es µA (40) = 0.45 en el conjunto difuso A.

Este valor de pertenencia tiene que ser reflejado hacia el *CONSECUENTE* (figura 5. 7) de la regla, es decir hacia el conjunto B (*velocidad de ventilación alta*) para obtener la función de la cual al momento de *defuzzificar* proporcionará el valor óptimo de velocidad a la cual se debe mover el ventilador de acuerdo a la temperatura que registraron los sensores.

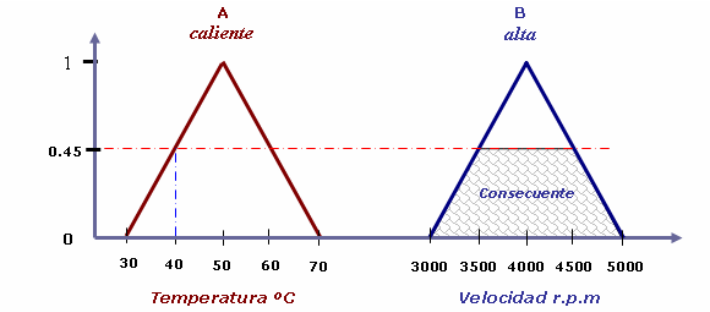


Figura 5.7 Representación del antecedente y consecuente de la regla heurística sobre el conjuntos difuso caliente y alta.

5.3.5 Desfuzzificación

La principal aplicación de los sistemas de razonamiento difuso es el control de dispositivos, que normalmente precisan de una salida nítida (acción de control). Así por ejemplo en el ejercicio anterior se requiere conocer cual es la cantidad óptima en revoluciones por minuto r.p.m en que debe girar el ventilador si la velocidad es alta. Existen diversas alternativas para transformar un valor difuso en uno nìtido (proceso que en inglés se llama defuzzification).

a) El valor máximo (es decir, el más posible). Si se producen empates puede seleccionarse el primer valor encontrado o la media (en cuyo caso el método se denomina media de máximos (MOM en FuzzyCLIPS).

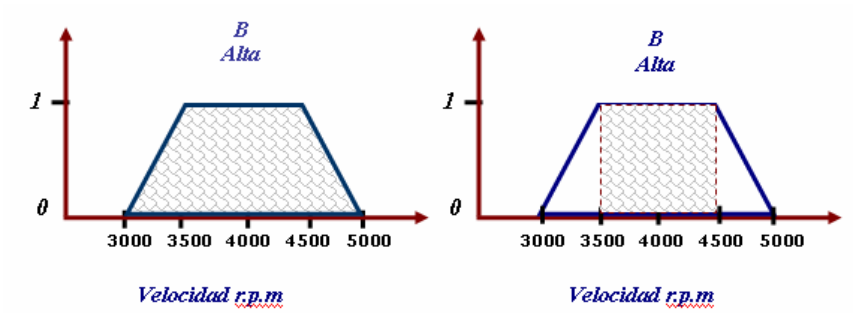


Figura 5.8 Desfuzzificación por medio del valor máximo.

En el ejemplo que se presenta, se encontró que la función de pertenencia tiene varios máximos: todos los valores entre 3500 y 4500. En la estrategia MOM se seleccionaría el valor medio, siendo éste, 4000, y la estrategia del primer valor máximo se seleccionaría 3500 (figura 5.8).

b) El centroide difuso (o centro de gravedad, COG en FuzzyCLIPS), definido como (figura 5.9):

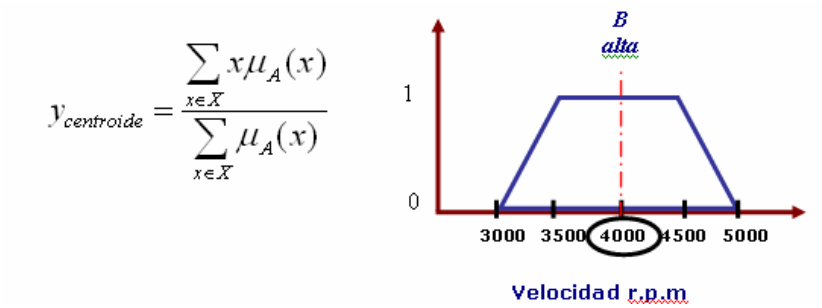


Figura 5.9 Desfuzzificación por medio del centroide.

Siendo:

$$\frac{(3000)(0)+(3500)(1)+(4000)(1)+(4500)(1)+(5000)(0)}{1+1+1} = 4000$$

*Obteniendo como **resultado** que la velocidad necesaria a la que debe girar el ventilador es de 4000 r.p.m, con una entrada de 45°C.*

Normalmente, si varias reglas tienen el mismo consecuente, lo que se suele hacer para acumular la evidencia es unir los dos conjuntos difusos resultantes y después defuzzificar el resultado.

5.4 Árbol de fallos difuso

5.4.1 Requisitos para trabajar con árboles de fallos difusos

La teoría de los conjuntos difusos se utiliza como herramienta para realizar análisis de fiabilidad al aplicarla en los árboles de fallos, los árboles de fallos difusos actualmente son empleados en la detección y diagnóstico de fallos en los componentes que integran al sistema por medio de observación de síntomas difusos usando la información contenida en el árbol de fallos.

Las primeras herramientas utilizadas para realizar análisis de la fiabilidad en robótica ha sido el árbol de fallos. Los árboles de fallos apoyan la codificación de la información del fallo de subsistemas extensamente dispares en un marco formal [Walke Ian., Carrera Carlos 2005].

El problema de la incertidumbre de los datos de entrada en el árbol de fallos afectan la exactitud (y por lo tanto la confianza) de las estimaciones de salida. La estructura de los árboles ha sido utilizada por varios autores para evitar este problema. Las desviaciones estándar de estadística en fiabilidad fueron utilizadas para limitar la desviación de estándar de la probabilidad del evento TOP en análisis del árbol de fallos. La representación de los datos de entada al árbol de fallos por medio de conjuntos difusos

permite cuantificar la incertidumbre definiendo funciones de pertenencia asociadas a éstos para designar el grado en que un elemento pertenece a un conjunto.

La teoría de la lógica difusa da una solución al problema de la representación del conocimiento y del razonamiento con datos imprecisos.

La idea de datos difusos se ha aplicado en el análisis de fiabilidad para los sistemas de control en los robots, donde cada entrada es tratada como conjunto de posibilidad para tratarla en la función pertenencia difusa que refleja la incertidumbre de los datos de entrada, dando como resultado un conjunto de datos que estiman la fiabilidad de salida. Sin embargo, según las indicaciones de [Carvallo, Walter, Leuschen 2004], éste método puede conducir a resultados extremadamente conservadores, y pueden dar lugar a la pérdida de incertidumbre en algunos casos.

Un valor difuso es una manera útil de representar la incertidumbre en los datos de entrada en el árbol de fallos. Una manera común definir un valor difuso es por medio de la función de pertenencia triangular con el vértice igual a $\mu(x) = 1$, el ancho de de base del triangulo son los valores que están dentro del conjunto difuso, y una posición apropiada en la base representa la incertidumbre, cuando se requiere representar una gran cantidad de datos, es preferible utilizar en el análisis funciones de pertenencia trapezoidales, a menudo este tipo de funciones son utilizadas para representar los porcentajes de fallos en sensores y actuadores de un robot [Leuschen M. 1993]. Las operaciones matemáticas para valores difusos se pueden definir utilizando el principio de extensión, éste principio generaliza la idea de función en conjuntos tradicionales, si f es una función de X a Y e A es un conjunto difuso sobre X definido como: $A= \mu_A(x_1)/x_1+\mu_A(x_2)/x_2 ++\mu_A(x_3)/x_3 +\ldots\ldots\ldots++\mu_A(x_n)/x_n.$

Entonces el principio de extensión que la imagen del conjunto A bajo la función f() se puede expresar como un conjunto difuso B, $B = f(A) = \mu_A(x_1)/y_1+\mu_A(x_2)/y_2 ++\mu_A(x_3)/y_3 +\ldots\ldots\ldots++\mu_A(x_n)/y_i$, el cual $Y_i = f(x_i) Y \mu_B(y) = max(\mu_B(x))$.

El principio de extensión proporciona los mecanismos matemáticos básicos para extender las expresiones matemáticas al dominio difuso, si X e Y son dos conjuntos nítidos (crisp sets) y f es un mapa de X a Y f: X $\rightarrow$ Y en el cual para cada x ? X, f(x) = y ? Y. Si A es un subconjunto difuso de X, se puede definir f(A) como un subconjunto difuso de Y en el cual $f(A) =? _x\{A(x)/f(x)\}$.

Ejemplo:

Si A es = {0.1/-2.0, 0.4/-1, 0.8/0, 0.9/1, 0.3/2}
y $f(x) = x^2-3$ aplicando el principio de extensión se tiene:

B = {0.1/1, 0.4/-2, 0.8/-3, 0.9/-2, 0.3/1}
= {0.8/-3,(0.4 ? 0.9)/-2,(0.1 ? 0.3)/1}
= {0.8/-3, 0.9/-2, 0.3/1 }

Sin embargo puede ser difícil de aplicar en la definición funcional de la función de pertenencia, además da como resultado una función de pertenencia no lineal. Este problema se puede evitar utilizando el alfa-corte (alfa-cut). Un corte-alfa es definido para ser el intervalo nítido (crisp), donde la función de pertenencia del un conjunto difuso es mayor o igual a un valor nítido, sobre el intervalo [0, 1], generalmente es más fácil aplicar el principio de extensión.

Alfa-corte a un nivel α intervalo real para cual todos los elementos pertenecen al número difuso con nivel de confianza mayor o igual a α , algebraicamente se tiene $\{x \in \Re / A(x) \geq \alpha\}$, soporte alfa-corte de nivel 0 , representado por. *A(0)*. Moda o uno-corte de nivel uno representado por A(1), figura 5.10.

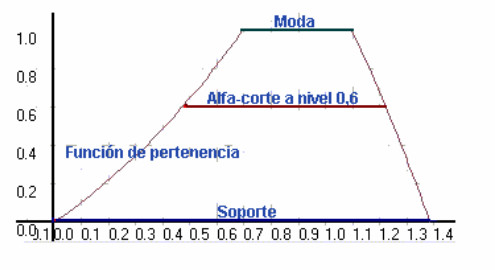


Figura 5.10 Soporte, alfa corte al nivel 0.6 y moda de un conjuto difuso.

Los análisis tradicionales con árbol de fallos utilizan la probabilidad para representar cada evento básico. Sin embargo, es poco realista evaluar la ocurrencia del evento TOP usando un valor nítido sin considerar la incertidumbre inherente e imprecisión que cada evento básico tiene. La lógica difusa se puede utilizar para ocuparse de esta clase de fenómeno. Un valor difuso se introduce para representar la probabilidad del fallo de un evento básico. El algoritmo del árbol de fallos difuso es desarrollado por el método del corte mínimo. La probabilidad difusa del fallo en un sensor o actuador del robot se puede calcular mediante el árbol de fallos difuso. Los eventos básicos principales que afectan la probabilidad de la ocurrencia del evento superior son verificados por el análisis de sensibilidad. Comparado el análisis tradicional con árbol de fallos y el método difuso implementado en árbol de fallos con éste último se puede conseguir la distribución de la posibilidad de la fallo en los sensores y actuadores de los robot, y así poder con mayor certeza diagnosticar sus posibles fallos.

La ventaja principal de usar modelos difusos en análisis de fiabilidad es que se preservan exactamente la incertidumbre a través del cálculo matemático que se realiza. La incertidumbre en la entrada debe ser propagada a través del modelo de tal manera que se pueda determinar la incertidumbre en la salida correctamente. Cierta cantidad de error es permitido, asumiendo que no se puede evitar y debe ser tolerado solamente si es más pequeño en magnitud que el ruido inherente en el modelo. A esto se le conoce para este modelo como criterio de incertidumbre (*uncertainty criterion*).

5.5 Análisis de fiabilidad con árbol de fallos difuso para los componentes que integran el sistema robótico

La lógica difusa se puede codificar fácilmente en un árbol de fallos para determinar la probabilidad global de la aparición de un fallo en un sensor, microcontrolador, actuador, tarea, etc., determinando el evento superior que define el fallo que puede tener el robot en los componentes antes mencionados, el objetivo principal de el análisis de fiabilidad con árboles de fallos difusos es propagar rangos de probabilidades de fallo en lugar de un solo valor que representa las probabilidades de los eventos básicos de árbol, este concepto introduce la difusidad en el análisis de fiabilidad mediante árboles de fallos difusos, donde los datos de entrada al árbol se representan por medio de una fusión de pertenencia difusa trapezoidal, que describe la posibilidad de la probabilidad del fallo de la entrada (figura 5.11), demuestra que toda probabilidad tiene una posibilidad de 1 sobre el rango de valores de p, a v. Entre u y v, y entre w y x, la posibilidad varía linealmente entre 0 y 1. Por lo que (u, v, w, x) describe

únicamente cada distribución. La idea de usar intervalos, o la probabilidad difusa depende de los datos disponibles. Por ejemplo, los datos de fiabilidad existentes para los motores eléctricos (usados típicamente en los robot móviles) se publica en las tablas de fiabilidad para componentes no electrónicos NPRD-95 da una probabilidad total de la fallo de 0.00924, junto con la información de que 68 por ciento de porcentaje de fallos estará entre 0.22 y 4.5 por el valor dado.

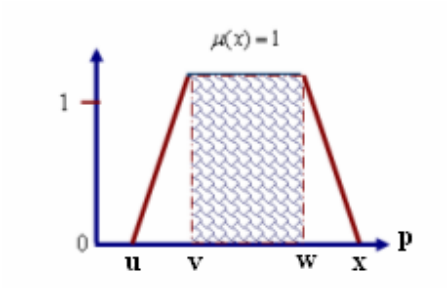


Figura 5.11 Representación de los datos de entrada difusos (posibilidad).

Por lo que se puede estimar, (u, v, w, x) como (0.0007392, 0.00203, 0.109032), para representar 90 por ciento será entre 0.08 y 11.9 veces el valor dado. Para el caso de un codificador óptico típico, la representación correspondiente de (0.00124, 0.00341, 0.0698, 0.184). Las estimaciones difusas antes mencionadas se pueden sumar y multiplicar usando las siguientes reglas:

Suma: $(u, v, w, x)_{nuevo} = (u_1 + u_2,\ v_1 + v_2, w_1 + w_2, x_1 + x_2)$
Multiplicación: $(u, v, w, x)_{nuevo} = (u_1 . u_2, v_1 . v_1, w_1 . w_2, x_1 . x_2)$

Las distribuciones de la entrada son dadas por (u_1, v_1, w_1, x_1) y (u_2, v_2, w_2, x_2).

El uso de estas reglas permite la combinación de tales estimaciones difusas a través de las puertas lógicas del árbol de fallos (AND, OR, etc.), mientras se preserva la esencia de la estructura difusa (las estimaciones están calculadas por parejas tomando en cuenta la función trapezoidal). Ese método, que se denomina acercamiento difuso, fue utilizado [Walker I.D., Cavallaro J.R. 1996] para evaluar los efectos de incertidumbre en el análisis de fiabilidad de todos los codificadores y motores que integran al robot. Por ejemplo, dado las distribuciones numéricas para el motor eléctrico y los codificadores ópticos, y si se dispara el evento E, la rueda R_i presenta un fallo, su probabilidad se da por $s_{iA} * s_{iB} + m_i$. Por lo tanto, si las probabilidades de entrada son independientes, la salida del trapezoide para el evento E siendo: E(u, v, w, x)=(0.0007407, 0.000204,0.04645,0.14289)

Este resultado refleja la incertidumbre en la salida, más allá de un único valor obtenido como resultado del análisis cuantitativo en el árbol de fallos, según [Walker I.D., Cavallaro J.R. 1996]. Sin embargo, las operaciones difusas utilizadas para generar el trapezoide difuso producen una estimación muy conservadora. Además, tales manipulaciones difusas tienen un límite que se explica hasta que las probabilidades de entrada varían con tiempo (es decir considerando el envejecimiento del sistema).

En este ejemplo se puede ver que según los datos de salida que el error es cero si se estiman intervalos en los datos de entrada Según lo definido aquí, esto significa la pérdida completa de incertidumbre.

La tabal 5.2 muestra la [Leuschen Martin 1997] columna de corte-cero y corte-uno describen la función trapezoidal difusa del porcentaje de fallo del valor normal. Se puede observar que con los resultados difusos se tiene un rango de posibles probabilidades de fallos. La incertidumbre en los porcentajes de fallos de los componentes es directamente proporcional en la incertidumbre del porcentaje de fallos en el sistema, estos cálculos también demuestran que se requiere redundancia en los motores para altos porcentajes de fallos. Con estos datos el diseñador puede evaluar que componentes principalmente sensores y actuadores debe duplicar sin cargar de elementos al robot. Tomando en cuenta los dados anteriores se decidió realizar el diseño de la arquitectura física del sistema de control para un robot móvil propuesta en este trabajo, que le permite al robot tolerar los fallos de una manera inherente, ya que es distribuido y cada nodo tiene su propio microcontrolador, si un nodo falla otro nodo puede soportar la carga de sus componentes de entrada y salida al tenerlos en doble conexión.

Componente del robot	*Valor normal*	*α-corte-cero*	*α-corte-uno*
Motor Eléctrico	***0.00924***	***[0.000739,0.11]***	***[0.00203, 0.0416]***
Sensor ultrasónico	***0.0123***	***[0.00102,0.145]***	***[0.00225,0.0498]***
Sensor infrarrojo	***0.0145***	***[0.00118,0.0170]***	***[0.00287,0.0567]***
Sensor codificador óptico	***0.0155***	***[0.00124,0.184]***	***[0.00341,0.0698]***
Sensor de batería	***0.00135***	***[0.00110,0.156]***	***[0.00245,0.0423]***
Sensor de contacto	***0.0129***	***[0.00104,0.139]***	***[0.00231,0.0504]***
Microcontrolador	***0.00034***	***[0.0000896,0.000186]***	***[0.000347,0.00481]***

Tabla 5.2 Porciento de fallo en los componentes del robot y su transformación difusa.

A continuación se dan ejemplos del uso de los árboles de fallos estándar para la localización y diagnóstico de los fallos en diferentes componentes de un robot.

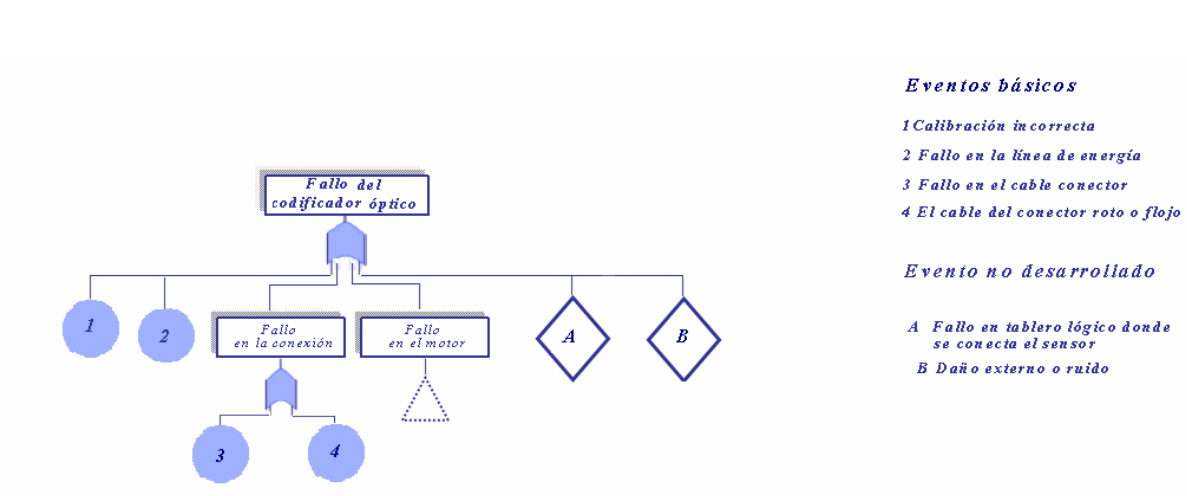


Figura 5.12 Análisis del modo de fallo con árbol de fallos para el codificador óptico del sistema robótico.

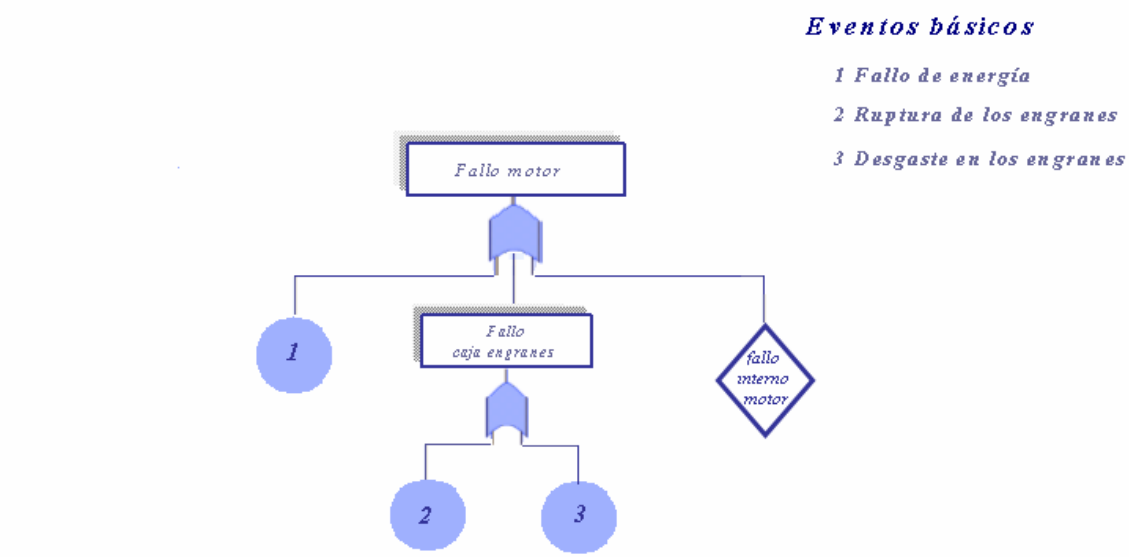


Figura 5.13 Análisis del modo de fallo con árbol de fallos para el motor eléctrico del sistema robótico.

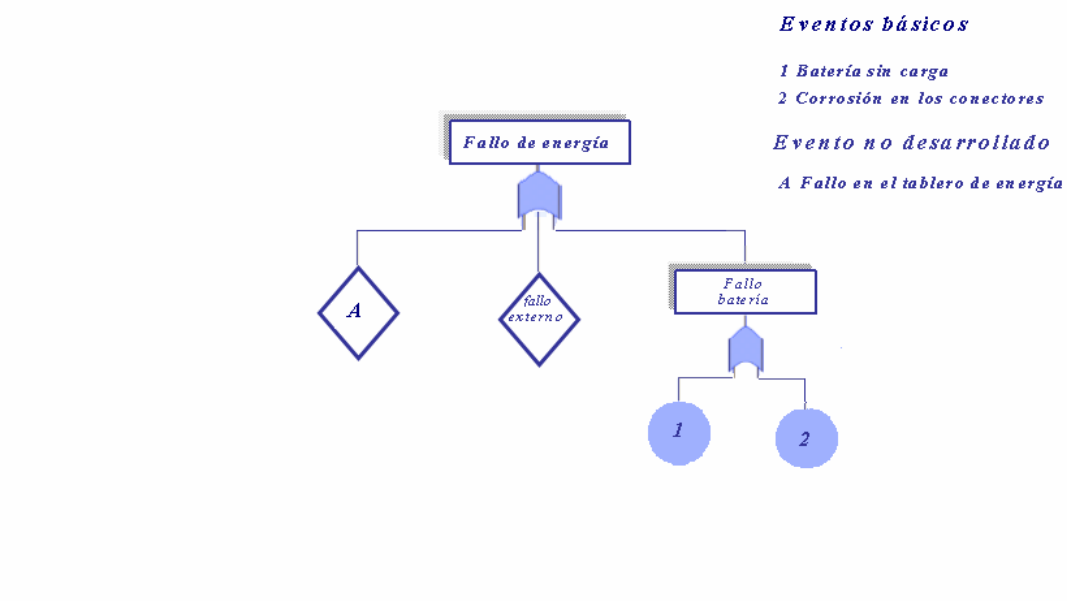


Figura 5.14 Análisis de modo de fallos con de árbol de fallos en el suministro de energía del robot.

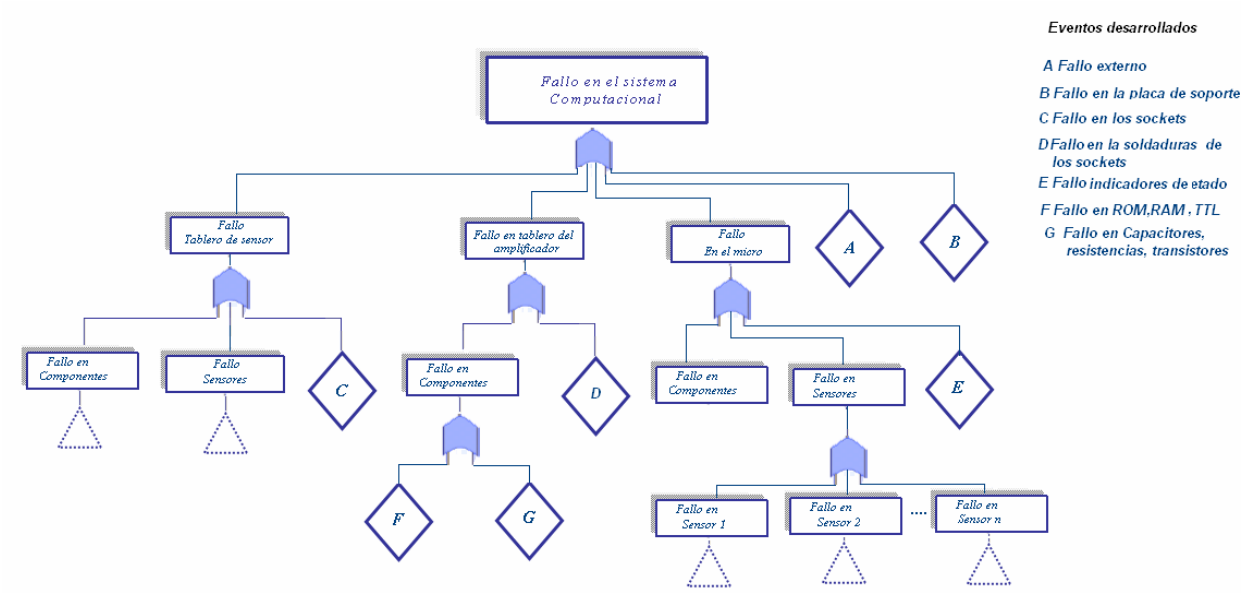


Figura 5.15 Análisis del modo de fallo con árbol de fallos del sistema computación del robot.

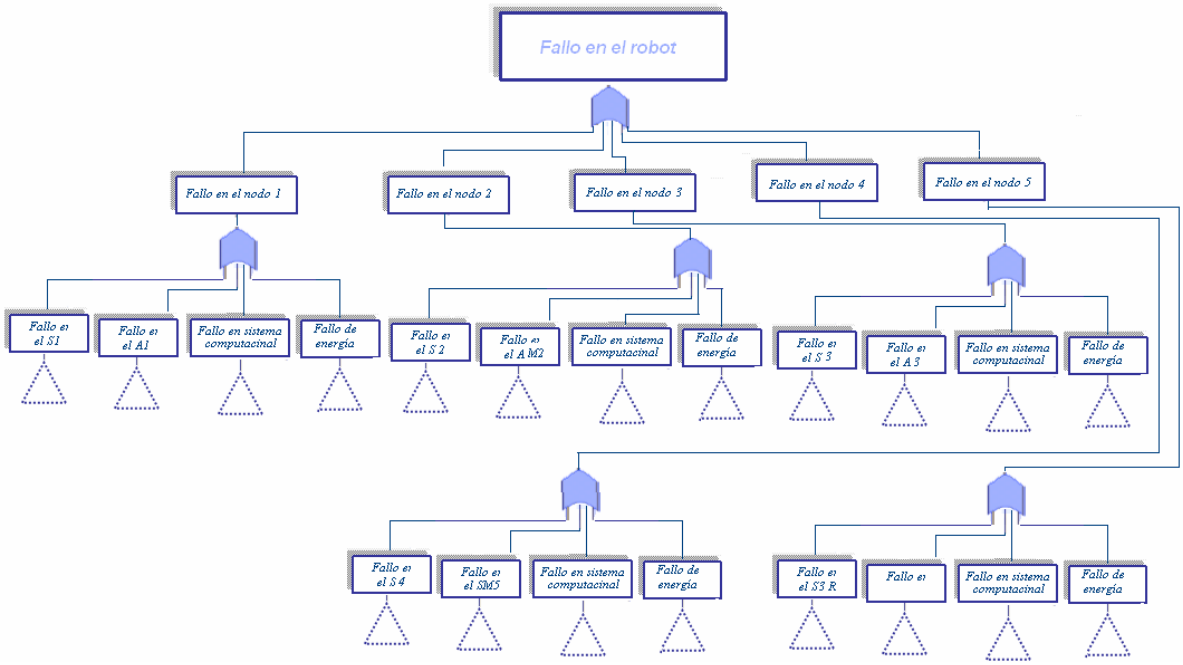


Figura 5.16 Análisis del modo de fallo con árbol de fallos del sistema de control propuesto para un robot móvil.

El método difuso de árboles e fallos es una técnica viable para analizar diseños tolerantes a fallos, es complejo ya que proporciona excesiva información y además mantiene la inherente difusidad del evento, es una herramienta ya establecida en fiabilidad y trabaja bien, su debilidad es que si no se calculan bien los intervalos de los datos de entrada en los eventos básicos del árbol de fallos se puede perder la incertidumbre.

En los árboles presentados en esta sección se puede apreciar que se analiza el modo de fallo de los componentes, de tal manera que los algoritmos de detección de fallos pueden ser fáciles de elaborar tomando la información contenida en el árbol.

5.6 Conclusiones del capítulo

Los modelos actuales de análisis de fiabilidad para el diseño de control de un robot móvil preferentemente, casi no se han realizado. Los trabajos investigados se enfocan primordialmente a robot estáticos, y los modelos existentes se basan primordialmente en modelos probabilísticos que comúnmente es inadecuado para esta área de conocimiento, ya que no se cuenta con suficiente información probabilística cuando se diseñan estos sistemas. La utilización de árboles de fallos difusos para realizar el análisis de fiabilidad de los componentes más vulnerables que integran a un robot móvil, nos preserva la información con la que se cuenta, que inherentemente es incierta. A pesar de la incertidumbre nos da un análisis, podemos confiar plenamente y diagnosticar los fallos en componente usando la información contenida en un árbol de fallos.

Capítulo 6 Modelo de la arquitectura tolerante a fallos mediante un SMA

6.1 Introducción

El primer paso que se realiza en el desarrollo de un sistema computacional es la definición de los requerimientos del problema al que se trata de dar solución, esto nos indica la complejidad que puede o no tener el sistema y las características que debe cumplir la metodología a emplear para su desarrollo. El propósito general de este trabajo es desarrollar formalmente un sistema basado en múltiples agentes que tiene todos los problemas de la distribución tradicional y de los sistemas concurrentes más la adición de dificultades que surgen de los requerimientos de flexibilidad e interacción sofisticada. Se llegó a la conclusión de que es imperante utilizar una metodología de desarrollo formal, por lo cual requerimos seleccionar una metodología conocerla para así poderla manejar para el proceso de modelado de nuestro sistema, y así podemos estar seguros que el modelo funcionará después de ser implementado y probado de acuerdo con los requisitos iniciales.

La metodología seleccionada MaSE cubre completamente el ciclo de vida del software, iniciando por la especificación inicial por medio de una frase, y produce un conjunto de documentos formales de diseño en estilo gráfico basado en una sintaxis formal. El producto final de MaSE es un diagrama que describe el despliegue de un sistema de agentes que se comunican a través de estructuras de conversación, esta metodología por medio de su herramienta de desarrollo en su versión profesional despliega el código de los de las clases de agentes.

Este trabajo se ha enfocado en el análisis y diseño del Sistema Multiagente tolerante a fallos en un sistema de control distribuido para un robot móvil (SCDRM) con las características expuestas en el capítulo anterior. Durante el desarrollo de éste capítulo se va describiendo el trabajo que se realizó en cada una de las etapas de la metodología seleccionada, con la finalidad de poder desarrollar nuestro Sistema Multiagente adecuadamente y eficientemente, por lo cual fue necesario describir cada etapa que conforma la metodología, y como la fuimos utilizamos para realizar el análisis y diseño del sistema propuesto en éste trabajo de investigación, es importante señalar que no se presenta todo el desarrollo del sistema propuesto en éste capítulo. Sin embargo el modelo completo se encuentra en el CD anexo a éste trabajo, asimismo como el simulador que dio origen después de realizar el modelo.

En la sección 6.2 se modela el SMA la fase de análisis en sus tres etapas que son: captura de metas, aplicación de casos de meta y refinamiento de roles. En la sección 6.3 se modela el SMA la fase de diseño en sus cuatro etapas que son: creación de las clases de agentes, construcción de las conversaciones, ensamblado de las clases de agentes y diseño del sistema. En la sección 6.4 se realiza la transformación de la fase de análisis a la fase de diseño. En la sección 6.5 se hacen las verificaciones de las conversaciones entre los agentes. La sección 6.6 se dan las conclusiones del capítulo.

6.2 Modelando el Sistema Multiagente propuesto en su fase de análisis

6.2.1 Introducción a la metodología MaSE

El principal enfoque de la metodología MaSE es ayudar al diseñador a tomar las especificaciones iniciales del sistema basado en agentes y producir el código de dicho sistema. Esta metodología cuenta con su herramienta de desarrollo denominada AgentTool diseñada y desarrollada por un conjunto de investigadores del Instituto de Tecnología de la Fuerza Aérea (AFIT) en su base de Ohio de los Estados Unidos Americanos, esta herramienta de desarrollo para MaSE también nos sirve como plataforma de validación y prueba del Sistema Multiagente a desarrollar.

A continuación se describen brevemente los siete pasos de los que consiste MaSE;

Ahora bien, la metodología permite dar seguimiento a los cambios a través del proceso ya que su ciclo de vida es tipo cascada. Cada diseño se puede rastrear hacia atrás o hacia delante a través de las diferentes fases de la metodología y sus correspondientes conceptos. De esta manera, se puede dar seguimiento a la inversa para encontrar los requerimientos iniciales que apoyan a un agente en particular. Además se puede hacer lo opuesto, un objeto de una fase inicial como una meta se puede rastrear a un conjunto de objetos de una fase posterior. El propósito es eventualmente poder seleccionar un objeto de diseño en AgentTool y recibir una retroalimentación visual de todos los demás objetos y sus efectos. La metodología MaSE consiste en siete pasos representados en la figura 6.1. Los rectángulos representan los diversos modelos empleados en los pasos y las flechas indican el flujo de la información entre los modelos. Los primeros tres pasos representan la fase de análisis de la metodología, mientras que los últimos cuatro pasos representan la fase del diseño.

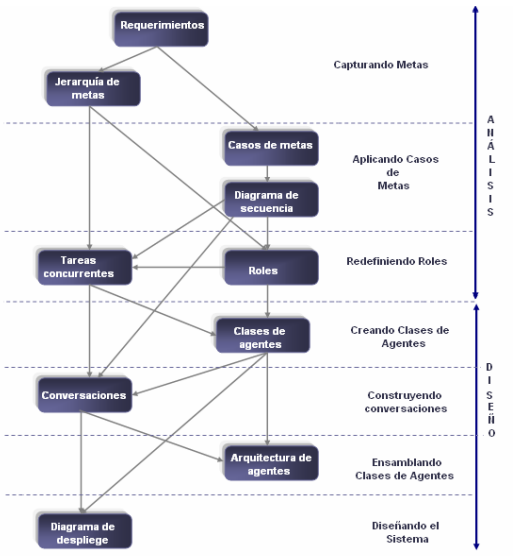


Figura 6.1 Fases de MaSE.

El primer paso en MaSE es la *captura de metas*, que toma las especificaciones iniciales del sistema y las transforma en un conjunto estructurado de metas que el sistema debe cumplir para funcionar de acuerdo al problema que debe resolver.

Esta fase es basada en las metas porque son una estructura altamente estable para construir el diseño del sistema. El procedimiento general de la fase de *captura de metas* es capturar lo que es importante a partir de los requerimientos de una manera estructurada y modular. La metodología MaSE utiliza metas como requerimientos de encapsulamiento debido a que las metas incluyen lo que el sistema trata de lograr y generalmente permanecen constantes por el resto del proceso del análisis y diseño. Esto se hace en contraste con otros tipos de análisis que están organizados alrededor de requerimientos que detallan como se debe hacer algo, como son los casos de meta. En esos casos los detalles suelen ser abrumadores y cambian con frecuencia [Kendall, Zhao 1998].

6.2.2 La captura de metas

En esta metodología, una *meta* siempre se define como un objetivo a nivel sistema, las construcciones de nivel inferior pueden heredar o ser responsables de las metas. En consecuencia, cada acción dentro de un sistema debe ser apoyada por una meta en particular. Una meta se expresa siempre como una declaración de lo que intenta el sistema, que inicia con las palabras: *El sistema deberá de*…Las metas se identifican extrayéndolas de la esencia del conjunto de requerimientos del sistema. Estos requerimientos pueden incluir documentos técnicos, relatos de los usuarios, o especificaciones formalizadas. Esto se inicia extrayendo los posibles escenarios a partir de la especificación inicial y describiendo la meta para cada escenario. Los requerimientos del sistema tolerante a fallos para el control de un robot móvil se describieron en el capítulo 4, de tal manera que la meta general es *tolerar los fallos en el sistema de control distribuido de un robot móvil a nivel hardware y software*. Para determinar el propósito de cada escenario se tuvieron que listar los requerimientos que identifican las metas. En la tabla 6.1 se enlistan una serie de metas que requiere nuestro sistema.

Tolerar los fallos en el sistema de control distribuido del robot móvil a nivel Hardware y Software.
✓ Detectar fallos en los actuadores que integran el sistema de control en el robot móvil.
✓ Detectar fallos en los sensores que integran el sistema de control en el robot móvil.
✓ Detectar fallos en las tareas del sistema de control.
✓ Detectar fallos en los nodos que integran al sistema distribuido de control del robot móvil.
✓ Aislar los sensores, actuadotes o tareas que presentan un fallo.
✓ Recuperar al dispositivo, tarea o nodo que tuvieron un fallo.
✓ Reconfigurar al sistema del fallo ocurrido en un dispositivo, tarea o nodo.

Tabla 6. 1 Requerimientos generales que identifican las metas de los escenarios del sistema de control obtenidos de los escenarios.

El paso final de esta fase es estructurar las metas analizándolas por importancia, y construyendo un *Diagrama Jerárquico de Metas*. Las metas más importantes deben ser reconocidas, y deben ser colocadas en la parte más alta del mencionado diagrama.

Las metas se deben estructurar de tal manera que se puedan distinguir entre las principales secuencias de interacción y los detalles subordinados. La característica importante que la jerarquía debe mantener es que todas los sub-metas se relacionen funcionalmente con su padre. En otras palabras, estas se relacionan con las mismas funciones o modos de operación de sus padres, pero en un nivel inferior. En algunos casos, las sub-metas también pueden subdividir lógicamente a sus padres.

Para el SMA que tolera fallos en el SCDRM, cuando se inició la captura de metas no se tenían claras algunas de ellas, por lo que se tuvo que definir en primer lugar de una manera clara y precisa la meta principal que es: *Tolerar los fallos en el sistema de control del robot móvil a nivel hardware y software.* Esta meta principal se optó por dividirla en dos sub-metas: tolerar fallos a nivel hardware y tolerar fallos a nivel software.

Al terminar con una sub-meta se prosiguió a realizar el análisis de la siguiente sub-meta ejemplo: *Tolerar fallos a nivel hardware*, se siguió el procedimiento de arriba hacia abajo, preguntándonos que se debe hacer para alcanzar dicha meta, analizado los requerimientos y posibles escenarios se logró realizarlo para cada uno de los niveles más bajos en el *Diagrama Jerárquico de Metas.*

Al momento que se fueron estableciendo las metas de los niveles superiores se fue dando la pauta para identificar las sub-metas, con la ayuda de los requerimientos y se afirmaron cuando se implementaron los casos de meta. Para el caso de la meta *tolerar fallos a nivel hardware* se separaron en las metas *tolerar fallos en actuadores únicos, tolerar fallos en actuadores múltiples, tolerar fallos en sensores único y tolerar fallos en sensores múltiples*.

Dado que la metodología MaSE nos permite interactuar entre las diferentes etapas, al momento de que se desarrollaron los casos de meta nos permitió redefinir metas que no habíamos logrado identificar al inicio de nuestro análisis, incluso logramos esclarecer mejor nuestras metas, tal es el caso de las metas mencionadas anteriormente, para efectos de funcionalidad se optó por separarlas en dos sub-metas: *tolerar fallos en dispositivos únicos y tolerar fallos en dispositivos múltiples*.

La figura 6.2 muestra el Diagrama Jerárquico de Metas para el SMA que tolera los fallos en SCDRM.

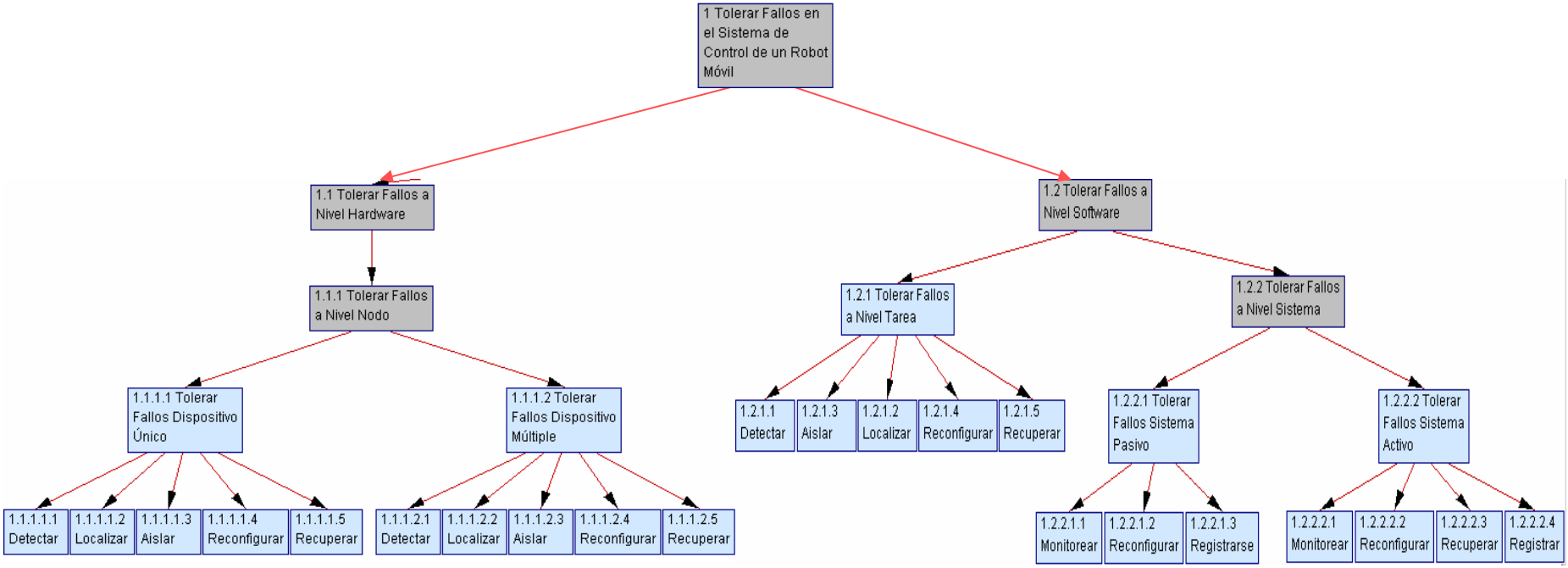


Figura 6.2 Diagrama Jerárquico de Metas realizado para SMA que tolera los fallos en el SCDRM.

El Diagrama Jerárquico de Metas es modular y nos permite modificaciones, cómo el agregar y borrar metas. Las flechas denotan las sub-metas de una meta en el nivel más bajo. La intención de cada nivel en el diagrama es contener pares de metas que tengan aproximadamente el mismo nivel de detalle. Éste diagrama aparenta ser una estructura de árbol pero no lo es realmente. Esto se debe a que metas idénticas se combinan para evitar redundancia, las cuales cruzan las ramas del árbol como puede apreciarse en la figura 6.3.

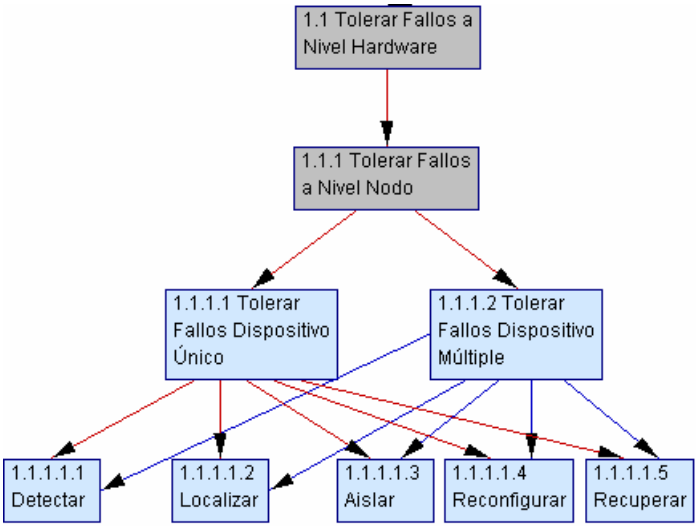


Figura 6.3 Diagrama Jerárquico de Metas eliminando redundancia.

En el diagrama mostrado en la figura 6.2 se visualiza claramente que las sub-metas del *Tolerador de Fallos en el Dispositivo Único* y *Tolerador Fallos en el Dispositivo Múltiple* contienen los mismos nombres de metas (Detectar, Localizar, Aislar, Reconfigurar, Recuperar), estas sub-metas pueden ser compartidas como lo muestra la figura 6.3, significando con esto que las tareas necesarias para lograr estas metas pueden estar contenidas en un solo rol; sin embargo, en nuestro diagrama jerárquico de metas se han puesto de manera separada ya que las tareas para lograr esas metas en cada Tolerador contienen diferentes estructuras en su funcionamiento, esto se puede ver claramente en los casos de meta y las tareas en el CD anexo.

La estructura mostrada en la figura 6.3 solo fue para ejemplificar el porque el diagrama de jerárquico de metas no es una estructura a manera de árbol. El diagrama jerárquico de metas también puede ser descrito mediante una lista en MaSE. Éste es un método alternativo para representar las metas (este listado no lo desarrolla la herramienta AgentTool). En nuestro caso presentamos un listado de los niveles más altos de la jerarquía de metas en tabla 6.2.

1. Permitir la tolerancia a nivel hardware y a nivel software para el sistema de control de un robot móvil.
    - 1.1. Tolerar fallos a nivel software.
        - 1.1.1. Tolerar los fallos a nivel sistema.
        - 1.1.2. Tolerar fallos a nivel tarea.
    - 1.2. Tolerar fallos a nivel hardware.
        - 1.2.1. Tolerar fallos a nivel nodo.
            - 1.2.1.1. Tolerar fallos a dispositivos únicos.
            - 1.2.1.2. Tolerar fallos a dispositivos múltiples.

Tabla 6.2 Listado de metas del diagrama de jerarquía de metas.

6.2.3 Aplicando casos de meta

Una vez que se capturaron todas las metas y se declararon explícitamente cada una de las metas identificadas, se cuenta con los cimientos para el modelo de análisis. Lo siguientes es identificar claramente los escenarios de los requerimientos, los cuales se detallan mediante los casos de metas.

6.2.3.1 Casos de meta para SMA que tolera los fallos en el sistema de control distribuido en un robot móvil (SCDRM)

El siguiente paso es la descripción y captura de los casos de meta que son las descripciones de una serie de eventos con una ocurrencia deseada en el sistema. Estos son ejemplos de cómo pensamos que el sistema debe comportarse en un momento dado.

Los casos de metas ya existen implícitamente cuando realizamos minuciosamente las especificaciones y los requerimientos del sistema, solo fue necesario identificarlos y extraerlos para describirlos de manera formal usando esta metodología.

Es importante en este punto del análisis del sistema no dejar que los escenarios y la creación de casos de metas se salgan de contexto y nunca se terminen. Cuando estábamos desarrollando nuestro sistema solamente fue necesario crear los suficientes casos de metas para cubrir las trayectorias potenciales de comunicación, y no todas las combinaciones posibles de los mensajes y datos. Esto se debe a la manera como los casos de metas los requeríamos utilizar, basándonos en recomendación que hacen los diseñadores de MaSE de que en cada secuencia de eventos del sistema que difieran significativamente de las otros se debe formar en un caso de meta. Una diferencia significativa puede ser un participante diferente involucrado en una secuencia de mensajes, una corriente de datos que va en dirección inversa, o cualquier objeto involucrado en un orden diferente a una secuencia de mensajes previos. Para lograr definir perfectamente en nuestro sistema los casos de metas nos orientamos, tomando en cuenta la declaración estándar *si, entonces* para cada caso de meta. Si sucede un evento A, el sistema debe realizar la acción B. De hecho, el sistema lo pudimos definir casi completamente con estas declaraciones de si, entonces, por caso de meta (aunque hacerlo fue un proceso exhaustivo). La tabla 6.3 muestra el escenario para lograr la tolerancia a fallos de un dispositivo múltiple, el cual pasó a ser un caso de meta.

Se desea saber como se tolerarán los fallos en un sensor múltiple, como es el caso de los sensores infrarrojos donde un robot siempre tiene varios para detectar obstáculos, en este caso pueden fallar un cierto número de sensores y el robot sigue trabajando, pero si llega a su tope el sensor ya es inútil para el robot. *De tal manera que el sistema tolerante a fallos debe ser capaz de buscar si se tiene un sensor de este tipo replicado, si existe debe de: reconfigurarlo de tal forma que el sistema siga trabajando sin degradarse, debe de ser capaz de aislar al sensor que falló, debe ser capaz de tratar de recuperarlo, si no se recupera, ponerlo como fallo permanente, si se recupera ponerlo como respaldo y seguir operando con la replica, debe ser capaz de actualizar su bitácora de fallos.* *Si el sensor no tiene replica, debe saber si este sensor no es crítico se debe de reconfigurar el sistema para que prosiga operando el robot en modo degradado, si es crítico se debe realizar un paro seguro.*

Tabla 6. 3 Escenario del SMA que tolera los fallos en el SCDRM.

El escenario anterior nos proporciona segmentos de información. Primero, nos ilustra una trayectoria de acción a través del sistema, segundo, nos introduce en algunos nuevos conceptos a la meta. En nuestro caso, el sistema de control se distribuye en nodos que controlan al robot, por lo que las tareas y dispositivos de entrada, salida, procesamiento y comunicación se distribuyen en los diferentes nodos. El escenario sugiere una secuencia de eventos, no obstante no los especifica, por lo que tuvimos que detallarlos cuando desarrollamos el caso de meta correspondiente, estos detalles de funcionamiento los tuvimos que declarar para cada caso de meta.

El caso de meta para el escenario anterior se presenta en el Apéndice A.

Los casos de metas nos resultaron muy valiosos dentro en la etapa de análisis de nuestro sistema ya que nos ayudó a las trazar trayectorias de comunicación que más tarde se convirtieron en conversaciones entre agentes. Teniendo esto en mente, fue necesario obtener suficientes casos de metas para cubrir las secuencias de eventos como nos fue posible. Por otro lado, no requerimos repetir una secuencia en particular varias veces con diferentes datos o tipos de mensaje ya que estas secuencias las usamos únicamente para determinar las trayectorias mínimas de comunicación.

Cuando estábamos desarrollando los casos de meta surgieron nuevas metas, y nos pudimos regresar fácilmente al paso anterior de esta metodología sin ningún problema y redefinir las metas.

6.2.3.2 Diagramas de secuencia para el SMA que tolera los fallos en el SCDRM

Al terminar de desarrollar los casos de metas teníamos la duda que si realmente estaban bien realizados, por tal motivo nos dimos a la tarea de diseñar unas tablas donde detallamos el funcionamiento de cada escenario, estas tablas se muestran en el Apéndice C, los casos de metas fueron comparados con las tablas, y si concordaban correctamente establecimos que el caso de meta estaba bien desarrollado de acuerdo a los requerimientos de nuestro sistema, de lo contrario hicimos un análisis más profundo hasta lograr que obtuvieran la concordancia. Cuando estábamos seguros que ya estaban bien realizados los casos de meta nos dedicamos a desarrollar los *Diagrama de secuencia*. El diagrama de secuencia, muestra la secuencia de eventos entre múltiples procesos. En MaSE, los diferentes procesos son los roles de agentes. A los eventos que se pasan entre los roles se les denomina mensajes.

Los diagramas de secuencia nos proporcionaron en nuestro análisis del sistema un panorama de alto nivel de cómo nuestros roles proceden recíprocamente para lograr las metas, y nos fueron de gran utilidad cuando construimos las tareas de cada uno de los roles que constituye nuestro sistema.

El diagrama de secuencia, nos fue útil para lograr determinar el mínimo movimiento de mensajes que existe entre nuestros roles. De tal manera que cuando se transmite un mensaje entre dos roles, fue necesario verificar que existiera una trayectoria correspondiente de comunicación entre ellos. Cuando al inicio obtuvimos un gran volumen de comunicaciones entre dos roles, la metodología nos facilito la manera de combínalos en una clase, tomando en cuenta lo que dicen los desarrolladores de MaSE que los agentes que dan origen estos roles heredan la comunicación entre ellos, y también afirman que una trayectoria de comunicación entre roles ejecutado por una clase diferente significa que una conversación debe existir entre las dos clases para que el mensaje ocurra.

Cuando se requirió transformar cada caso de meta a su diagrama de secuencia lo pudimos hacer de manera directa en AgentTools. Lo primero que realizamos fue la identificación de los roles del sistema y de sus eventos. Determinamos a cada participante (que es un rol) en cada diagrama de secuencia. Especificamos un diagrama de secuencia por cada caso de meta. La metodología permite definir varios diagramas de secuencia para un caso de meta si se detecta varias posibilidades de enviar un mensaje entre los mismos roles.

Después de identificar los roles que integran nuestro sistema el paso siguiente fue diseñar los diagramas de secuencia, para lograrlo fue necesario releer todo los casos de meta, y localizar todas las instancias de un evento que ocurran entre dos roles. Cada evento en cada caso de meta se dibujo con una flecha en el diagrama de secuencia en el orden en que ocurrieron.

Cuando aplicamos los casos de meta y logramos definir los diagramas de secuencia, todas las secuencias potenciales de eventos se tomaron en cuenta cuando tuvimos que diseñar las conversaciones que designamos en los casos de meta. Tomando en cuenta que los diagramas de secuencia operan entre los roles del sistema en esta metodología, al diseñarlos nos generaron las trayectorias de comunicaciones entre los roles que definimos.

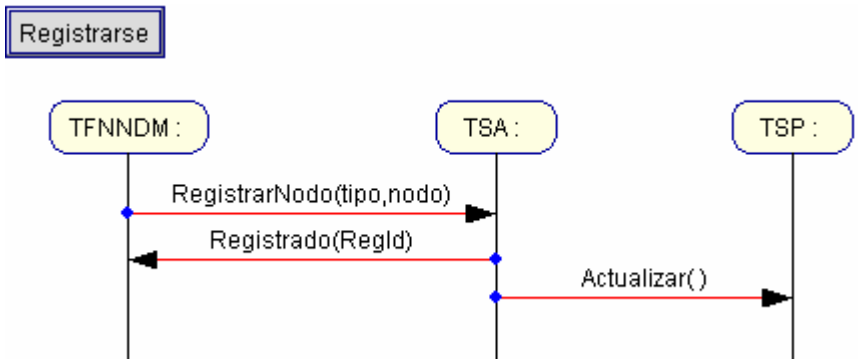


Figura 6.4 Diagrama de Secuencia para el caso de meta del registro de los agentes ante el rol TSA.

La figura 6.4 muestra el diagrama de secuencia que genera el caso de uso registrarse, realizado en la herramienta de desarrollo AgentTool y sustentado por metodología MaSE, el diagrama de secuencia representa una serie de eventos enviados entre los roles Tolerador de fallos a nivel Nodo del Dispositivo Múltiple (TFNNDM), Tolerador de Fallos a nivel Sistema Activo (TSA) y Tolerador de Fallos a nivel Sistema Pasivo (TSP), los cuales son requeridos para el registro del Tolerador de Fallos de un Dispositivo Múltiple. Estos eventos están contenidos en las conversaciones que desarrollamos en la etapa correspondiente de esta metodología.

6.2.4 Redefiniendo roles

El siguiente paso de la metodología en su fase de análisis, es transformar las metas estructuradas jerárquicamente en una forma más útil para construir el SMA, y estos son los antes mencionados roles.

Los roles son los bloques básicos para la construcción de los agentes en MaSE y representan las metas del sistema durante la fase de diseño. Al asociar cada meta con un rol, las metas serán cumplidas, ya que cada rol será ejecutado por una clase de agente.

Los roles son el fundamento del diseño en MaSE. Específicamente, son usados para construir clases de agentes. Más allá, los roles contienen una serie de metas definidas en la fase de análisis. En otras palabras, los roles forman un puente entre lo que el sistema quiere lograr y lo que va a hacer para lograrlo.

Cualquier participante en los casos de meta creados debe implicar un rol en apoyo a la meta involucrada en ese caso de meta. Un participante en un caso de meta es cualquier entidad que envía o recibe información como parte de una secuencia de eventos.

6.2.4.1 Transformar metas en roles

Nosotros transformamos las metas en roles uno a uno; cada meta la trazamos a un rol. Sin embargo, hay muchas situaciones donde es útil combinar metas. Las metas similares o relacionadas se pueden combinar en roles sencillos en búsqueda de conveniencia y eficiencia.

Los roles los detallamos en una lista (como lo especifica MaSE) como se muestran en la tabla 6.4. Los números dentro de los paréntesis indican las metas asociadas a cada rol del diagrama jerárquico de metas.

✓ Tolerar Fallos Dispositivo Único	(1.1.1.1, 1.1.1.1.1, 1.1.1.1.2, 1.1.1.3, o 1.1.1.1.5)
✓ Tolerar Fallos Dispositivo Múltiple	(1.1.1.2, 1.1.1.2.1, 1.1.1.2.2, 1.1.1.2.3, o 1.1.1.2.5)
✓ Tolerar Fallos Nivel Tarea	(1.2.1, 1.2.1.1, 1.2.1.2, 1.2.1.3, 1.2.1.4, o 1.2.1.5)
✓ Tolerar Fallos Sistema Pasivo	(1.2.2.1, 1.2.2.1.1,1.2.2.1.2,1.2.2.1.3)
✓ Tolerar Fallos Sistema Activo	(1.2.2.2, 1.2.2.2.1, 1.2.2.2.2, 1.2.2.2.3, o 1.2.2.2.4)
✓ Tolerador Dispositivo	(1.1.1.1.4, 1.1.1.2.4)
✓ Reconfigurador Sistema	(1.2.2.2.2, 1.2.2.2.3)

Tabla 6.4 Se listan los roles que conforman el SMA tolerante a fallos para el sistema de control distribuido en un robot móvil.

Los roles los identificamos por medio del diagrama de metas, y,de los casos de meta. Generalmente una meta padre de algún nivel es candidata a ser un rol, ya que las sub-metas están implicadas en el mismo objetivo; hay que tener en cuenta el grado de dificultad de las tareas que se encargarán de cumplir esas metas y sus relaciones (esto se aprecia más en los casos de meta), si varias metas contienen gran cantidad de interacción con otras metas por medio de las tareas es posible formar un rol. Para el SMA que tolera los fallos en SCDRM se obtuvieron siete roles, de los cuales los primeros cinco roles están contenidos en la tabla 6.4 se definieron del diagrama jerárquico de metas, y al obtener los participantes de los casos de meta. Respecto a los roles Tolerador_Disp y Reconfigurador_Sist se definieron al realizar las tareas de los primeros cinco roles, y al analizar los casos de meta; las tareas que desempeñan estos dos roles cumplen una misma meta contenida en dos casos de meta, por ejemplo el reconfigurar un dispositivo único o múltiple tiene el mismo comportamiento en el rol que soporta al dispositivo múltiple que el del dispositivo único al igual en el momento de registrar sus toleradores, por lo que se decidió poner en roles diferentes las tareas que contenían la misma funcionalidad.

La figura 6.5 muestra los roles identificados para nuestro sistema, estos roles son: TFNNDU rol que tolera los fallos en los dispositivo únicos, TFNNDM rol que tolera los fallos en los dispositivos múltiples, TFTarea rol que tolera los fallos en las diferentes tareas que constituyen

al sistema de control, TSA rol que tolera los fallos a nivel sistema de manera activa en el SCDRM y el rol TSP que tolera los fallos a nivel sistema inactivo contenido en cada nodo del SCDRM. El rol TSA tiene la función de registrar al tolerador de fallos del dispositivo único así cómo al múltiple, además de reconfigurar a los dispositivos en los nodos; el rol Reconfigurador_Sist reconfigura los nodos al momento de entrar a modo de fallo cualquier nodo. La figura 6.5 muestra a los diferentes roles que integran nuestro sistema, cada uno contiene el listado de las metas que debe realizar mediante las tareas y comunicaciones que se desarrollaran en las siguientes de MaSE.

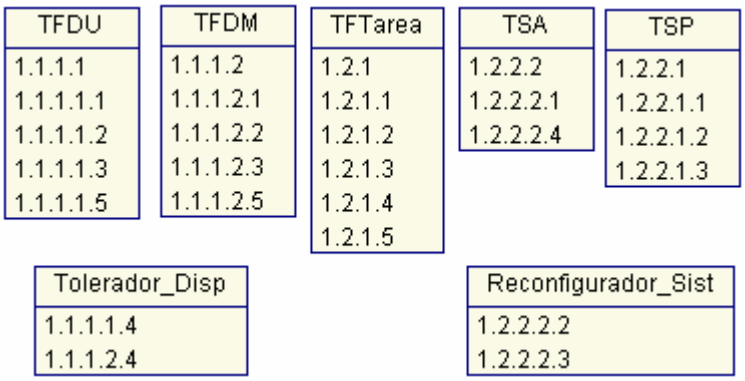


Figura 6.5 Roles que conforman el SMA que tolera los fallos en el SCDRM.

La colección de roles que constituyen nuestro SMA utilizando la metodología de desarrollo MaSE se le llama *modelo de roles*. Un modelo de roles es una abstracción para modelar y diseñar sistemas basados en redes de agentes.

Cuando se construyen los roles, se pueden etiquetar, y archivar; de tal manera que si realizamos un nuevo diseño de un Sistema Multiagente, estos roles los podemos adaptan al nuevo sistema reutilizándolos, aplicándolos al nuevo sistema ahorrando trabajo.

La herramienta de MaSE (AgentTool) nos provee esta conceptualización, ya que la organización y localización de roles dentro de las clases de agentes en análisis de un SMA lo podemos cambiar fácilmente, ya que los roles pueden ser manipulador por módulos con esta herramienta.

La figura 6.6 muestra el modelo de roles de perteneciente a nuestro SMA. Los rectángulos representan los roles y dentro de estos se localizan enumeradas las metas que cada rol debe cumplir de acuerdo a al diagrama jerárquico de metas.

Esto nos permitió asegurarnos de que todas las metas del diagrama jerárquico de metas se las hayamos asignado a un rol. A los roles les asignamos sus tareas concurrentes correspondientes, las listas que están contenidas en le Apéndice C nos ayudaron a definir correctamente cada tarea que debe desempeñar cada rol para cumplir las metas, las tareas en el diagrama son representadas por un óvalo unido al rol.

Las líneas entre las tareas denotan los protocolos de comunicación que ocurren entre estas. Las flechas indican que tarea es el iniciador y cual tarea es el respondedor en el protocolo.

Las flechas de color rojo en línea continua indica una comunicación externa entre dos tareas de diferentes roles o entre dos tareas de diferentes instancias del mismo rol.

Los protocolos externos involucran mensajes que se están pasando entre roles y se convertirán en mensajes en una conversación entre las clases de agentes que desempeñan estos roles.

Las líneas de color azul punteadas denotan comunicación entre dos tareas que pertenecen a la misma instancia del rol.

Se diseñaron roles que comparten una misma tarea. En MaSE se pueden duplicar tareas Si se determina que dos roles deben tener la misma tarea, entonces los roles necesitan una descomposición adicional. Las tareas compartidas se deben poner bajo un rol separado, y esos roles se pueden volver a combinar en una sola clase del agente en la fase del diseño, en nuestro caso no duplicamos tareas.

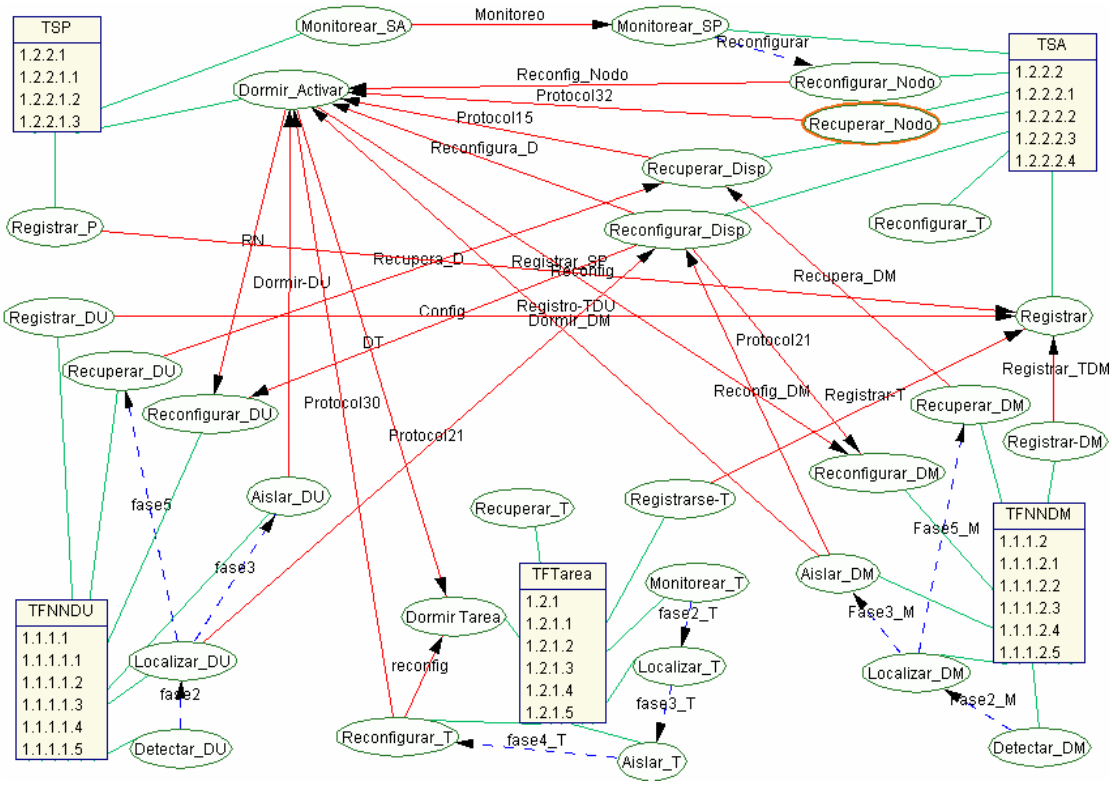


Figura 6.6 Modelo de Roles para el SMA que tolera los fallos en el SCDRM.

La figura 6.7 muestra una representación del modelo de roles del caso de meta del Tolerador de Fallos a Nivel Nodo en un Dispositivo Único (la implementación de su caso de meta se muestra detalladamente en el Apéndice B); en esta figura se puede observar los roles involucrados para este caso de meta y la distribución de las tareas entre los roles necesarias para realizar la secuencia de eventos descritas en el caso de meta. De igual manera se observan los protocolos requeridos para la interacción entre las tareas.

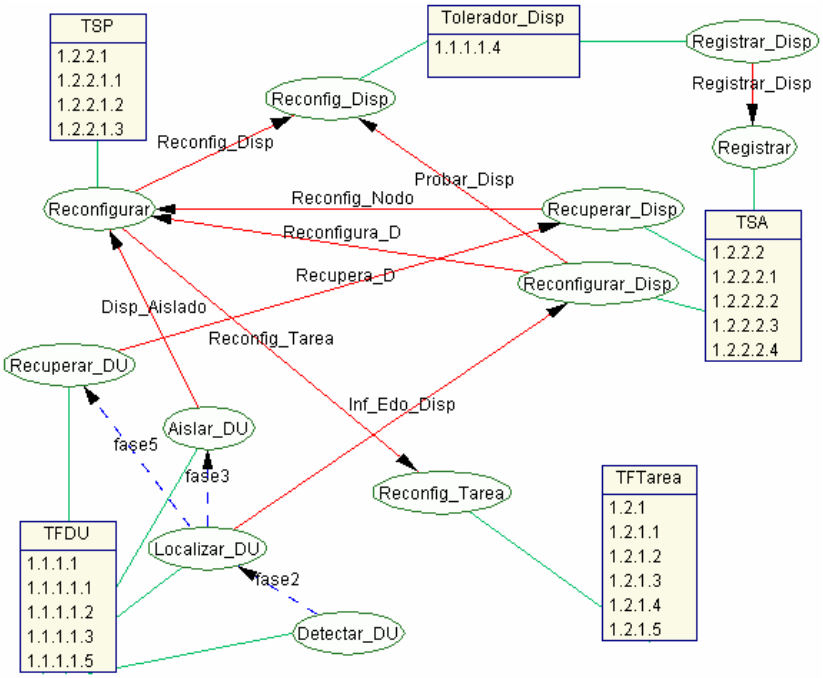


Figura 6. 7 Relación entre los roles para el caso de meta

Los roles se componen de las metas definidas en la primera parte del análisis, es importante definir quién realizará el proceso operativo para que se cumplan estas metas y defina el comportamiento de los agentes que serán realmente representados por los roles, este proceso es realizado por las tareas concurrentes que son una pieza importante cuando se está desarrollando un SMA utilizando la metodología MaSE.

En la siguiente sección describimos la estructura y la manera en que se realizaron las tareas concurrentes para nuestro sistema. Así mismo se especifica detalladamente a los componentes de las tareas concurrentes, tomando en cuenta que las tareas son las bases de los agentes, de las cuales surgen los componentes del agente y las conversaciones.

6.2.4.2 Tareas concurrentes

Después de que desarrollaron los roles y que pudimos definir las meta asociada a cada uno de los roles, el siguiente paso fue determinar las tareas concurrentes que debe de ejecutar cada rol para poder cumplir sus metas, el objetivo de usar modelos de tareas concurrentes en esta metodología, es que ayuda a definir el comportamiento interno de los agentes, y definen las interacciones con otros agentes relacionado a estos procesos internos.

Las tareas contienen un solo hilo de control que define una tarea que el agente puede hacer e integra interacciones dentro del agente así como también con otros agentes.

Las tareas concurrentes están contenidas gráficamente empleando un autómata de estados finito como se puede ver en la figura 6.8. Se asume que todas las tareas empiezan su ejecución al momento del inicio del agente y continúan hasta que el agente termina o se llega al estado de fin.

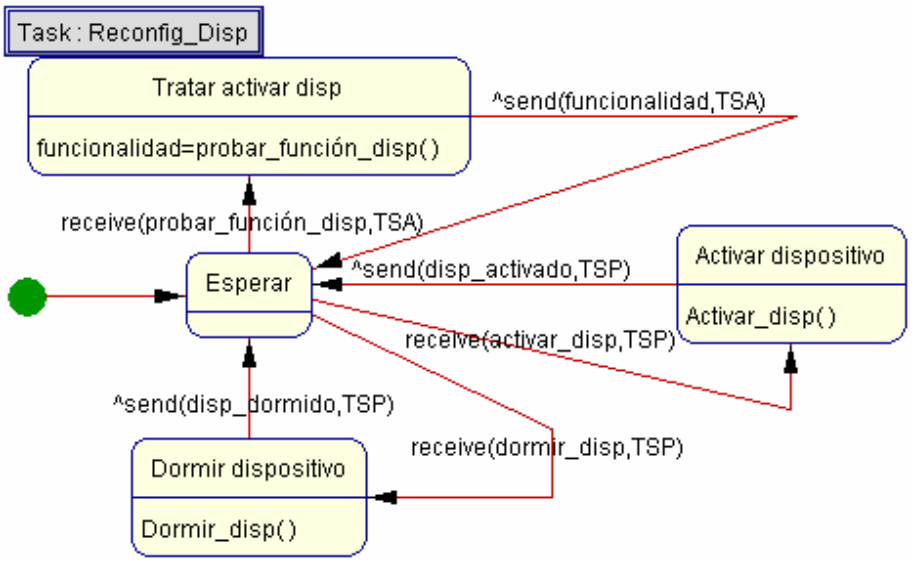


Figura 6. 8 Tarea concurrente Reconfig_disp en el SMA que tolera los fallos en el SCDRM.

El comportamiento del agente se modela con un número ***n*** de tareas concurrentes ejecutadas en paralelo. Las actividades son usadas para especificar las funciones reales llevadas a cabo por el agente que son definidas dentro de los estados del agente. Mientras que estas tareas se ejecutan concurrentemente y llevan a cabo comportamiento de alto nivel, pueden ser coordinadas usando eventos internos. Los eventos internos van de una tarea a otra y están contenidos en las transiciones entre estados. Para comunicarse con otros agentes, se envían y reciben mensajes internos llamados eventos internos *send* y *receive*. Estos eventos envían y recuperan mensajes del componente manejador de mensajes del agente, el cual debe existir. Además de la comunicación con otros agentes, las tareas pueden interactuar con el ambiente por medio de lecturas perceptivas o llevando a cabo operaciones que afecten el ambiente. Esta interacción es generalmente captada en funciones definidas en los estados. Por medio de incluir razonamiento dentro de las tareas, los agentes no son puramente reflexivos. Pueden planear, buscar, o usar el razonamiento basado en conocimiento para decidir acciones apropiadas.

La sintaxis de una tarea concurrente tiene dos componentes: *los estados y las transiciones.* Los estados abarcan el procesamiento que va interno al agente. Este procesamiento está denotado por una secuencia de actividades especificadas de una manera funcional. Las transiciones denotan comunicación entre los agentes o las tareas.

La figura 6.9 muestra la tarea Detectar_DM, la cual contiene dos estados que son: Monitorear-*DM y Cambiar Estado*, de igual manera contiene 5 transiciones:

1. Del estado inicio al estado Monitorear-DM, lo que indica que las funciones contenidas (Entrada_Error=Monitorear_Disp_Multiple()) en este estado se ejecutarán instantáneamente al momento de entrar en función el SCDRM.
2. Entrada_Error=LibredeError, indica que mientras no exista un fallo en algún dispositivo esta transacción regresa al estado Monitorear-DM, con lo cual se continua monitoreando al dispositivo múltiple.
3. La transición Monitorear del estado Monitorear-DM indica que es una transición proveniente de otra tarea que pertenece al mismo rol.
4. Entrada_Error=Error indica que al momento de existir algún posible fallo en uno de los dispositivos pasará al estado Cambiar Estado.
5. Por último la transición Localizar, la cual indica que se envía un evento (Localizar) a otra tarea contenida en el mismo rol.

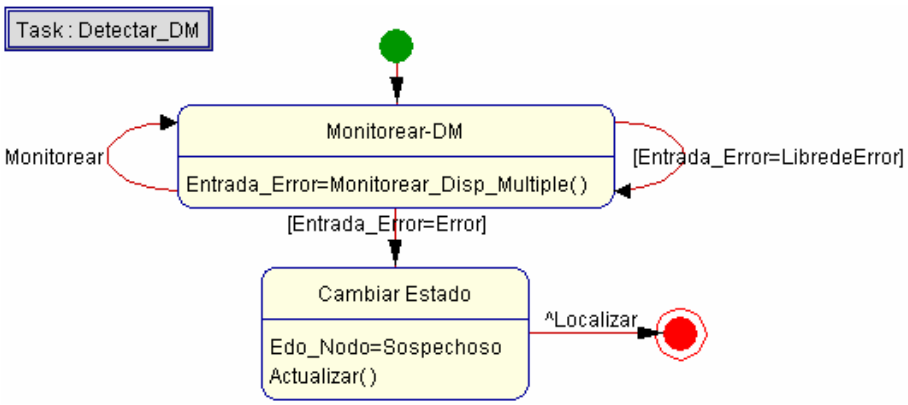


Figura 6.9 Descripción de la tarea concurrente del SMA que tolera los fallos en el SCDRM.

Transiciones

Una transición consiste de los siguientes elementos:

1. Cada transición tiene un solo estado fuente y un solo estado destino.
2. Una transición también tiene un disparador (*trigger)*, el cual puede ser un mensaje que envía un agente externo (un evento *receive*) o un evento interno que se envía desde otra tarea. Los parámetros del evento disparador son el mensaje y el agente del cual el mensaje fue enviado. Su sintaxis es: *receive (mensaje, agente)*. Un evento receive es solamente valido como un disparador (trigger).
3. Cada transición puede también tener un guardia (*guard*), es una condición booleana y tiene que ser verdadera antes de que pueda ocurrir la transición.
4. Una transición también consta de *transmisiones*, que son ya sea, un mensaje a un agente externo (un evento *send*) o bien, un evento enviado a otra tarea. En una transición pueden ocurrir múltiples transmisiones que son separadas con un punto y coma (;). El punto y coma no solo separa las transmisiones, sino que también imparte un ordenamiento secuencial a su transmisión. Su sintaxis es: *send(mensaje, agente).*

Las transiciones pueden ser hechas con sólo una condición guardia y sin disparadores o transmisiones. La sintaxis para una transición es: *Trigger [guard] / transmission(s).*

Generalmente, los eventos especificados en un disparador o las transmisiones se asume que vienen de otra tarea semejante dentro del mismo rol o agente. Esto le permite a un agente coordinar sus tareas. Sin embargo, dos eventos especiales son usados para indicar que un mensaje es realmente enviado del agente en uso a otro agente: *send* y *receive*.

El mensaje especificado en la sintaxis de estos eventos es definido como *performativa*, que describe la intención del mensaje, junto con un conjunto de parámetros que son el *contenido (content)* del mensaje. El formato de un mensaje es: *performativa* (p_1,p_2,p_3,…. p_n); donde p_1,p_2 p_3…. p_n denota *n* posibles parámetros. Es posible que un mensaje contenga solo una performativa sin contenidos. En este caso, solo la performativa y el identificador del agente son requeridos. Por ejemplo, para enviar un mensaje de reconocimiento, el mensaje tendría la sintaxis siguiente: ***send (acknowledge, agent)***, donde *acknowledge* es la performativa y el segundo parámetro es interpretado como el agente al que se le envía el mensaje.

Es posible también poder enviar un mensaje a un grupo de agentes por medio de *multicasting*. Esta es una capacidad común soportada por muchas estructuras de comunicaciones de agentes y pueden ser simuladas por medio del envío de mensajes múltiples, si no es directamente soportada por la estructura de comunicación. En vez de determinar un solo agente al cual enviar un mensaje (*message)*, se especifica un nombre de grupo (*group-name*) encerrando el nombre

del grupo con < >. La sintaxis para un mensaje multicast es: s*end(message, <group-name>).* En el caso de los eventos *send* y *receive*, debe siempre existir por lo menos un parámetro que denote el agente del cual el mensaje fue enviado o recibido. Una transición está habilitada si todas las siguientes condiciones se están cumpliendo

1. El estado origen de la transición es el estado actual de la tarea.
2. El evento *trigger* (disparador) de la transición ha sido generado.
3. La condición *guard* (guardia) de la transición se evalúa en verdadero.
4. Todas las actividades en el estado origen de la transición han sido completadas.

Si una transición no tiene un *trigger* o un *guard*, ambas condiciones se asume que se están cumpliendo. Una vez que una transición está habilitada, es ejecutada y la ejecución ocurre instantáneamente. Esto significa que los eventos y los mensajes son enviados instantáneamente y el estado de la tarea actual se vuelve el estado destino de la transición. En el caso de una transición que envía dos eventos o mensajes, aunque la transición ocurra instantáneamente, los eventos y mensajes son ordenados secuencialmente de acuerdo al ordenamiento en el diagrama. Si múltiples transiciones son habilitadas simultáneamente, el siguiente esquema de prioridad es usado.

1. Eventos recibidos. Cualquier transición cuyo *trigger* contenga un evento recibido de otras tareas es procesada primero. Si transiciones múltiples con eventos internos están habilitadas, entonces son procesadas en el orden que los eventos fueron recibidos. Debido a que los eventos son transmitidos instantáneamente, debe haber un ordenamiento lineal para los eventos.
2. Eventos enviados. Cualquier transición cuyas transmisiones contengan un evento para ser enviado a otra tarea es procesada enseguida. Si múltiples transiciones con eventos *send* están habilitadas, deben ser ordenadas en base a otro criterio tal como tiempo de llegada del *trigge*r, etc. Debido a que las condiciones *guard* deben ser mutualmente exclusivas y los eventos y mensajes *triggers* (disparadores) son ordenados en base a tiempos de llegada, siempre hay un ordenamiento al envío de eventos.
3. Mensajes recibidos (*receive*). Cualquier transición cuyo *trigger* contenga un mensaje recibido de otro agente es procesada enseguida. Si múltiples transiciones con mensajes recibidos están habilitadas, entonces son procesadas en el orden en que los mensajes fueron recibidos. Debido a que los mensajes, como eventos, son transmitidos instantáneamente, debe haber un ordenamiento lineal para el reconocimiento de los mensajes.
4. Mensajes enviados (*send*). Cualquier transición cuyas transmisiones contengan un mensaje para ser enviado a otro agente es procesada enseguida. Si múltiples transiciones con mensajes *send* están habilitadas, deben ser ordenadas en base a otro criterio, tal como el tiempo de llegada del disparador, etc. Debido a que las condiciones *guard* deben ser mutualmente exclusivas son ordenados en base tiempos de llegada, siempre hay un ordenamiento al envío de eventos.
5. Condiciones *guard.* Las transiciones que sólo tienen condiciones *guard* son las siguientes en prioridad. La regla para condiciones *guard* es que los estados que contengan múltiples transiciones de condiciones *guard* deben ser mutualmente exclusivas. Puesto que los eventos recibidos y los mensajes son una parte implicada de una condición *guard* de una transición, estás deben estar en conjunción con la condición guard cuando se está determinando la exclusividad.
6. Transiciones *null* (nula). La única transición *null* (nula) es una transición desde el estado inicial o *start*. Si hay una transición nula del estado *start*, no habrá otras transiciones del estado *star*.

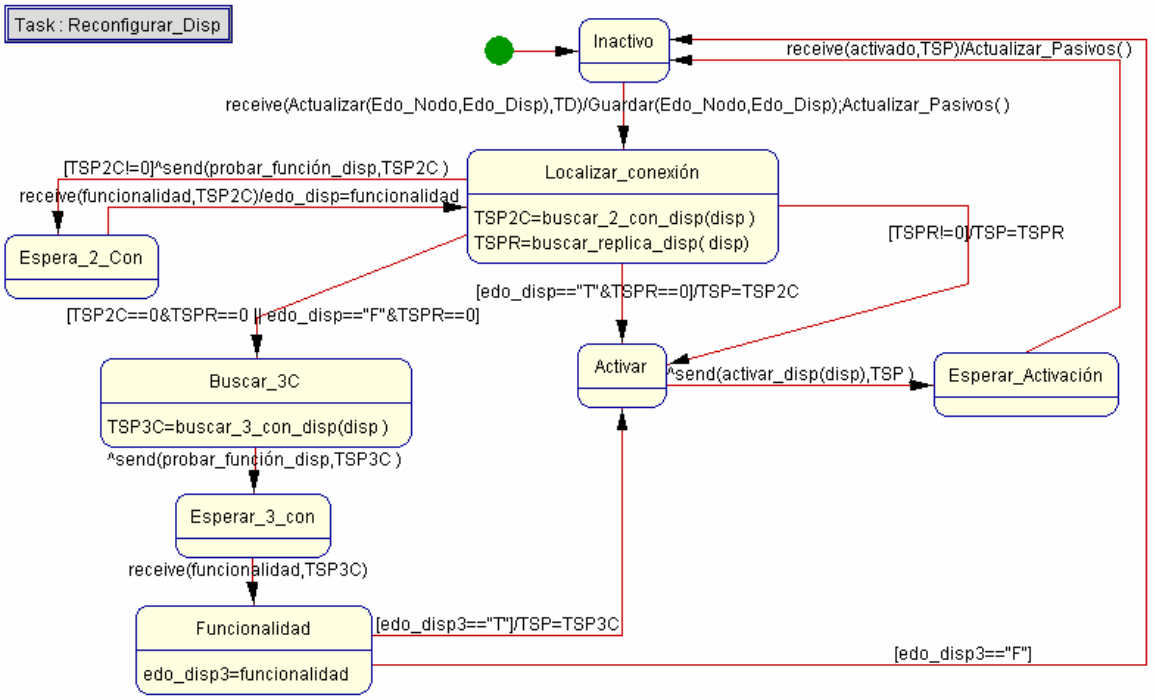


Figura 6.10 Diseño de la tarea concurrente Reconfigurar_Disp para el SMA del rol TSA.

En la figura 6.10 se muestra la tarea Reconfigurar_Disp perteneciente al rol TSA, esta tarea permite al rol tolerador de fallos a nivel sistema activo realizar las acciones pertinentes para reconfigurar un dispositivo (único o múltiple) al momento de fallar en algún nodo. Lo primero que trata de realizar es activar al dispositivo si este tiene una replica o no.

La transición entre el estado Inactivo al estado Localizar_Conexión consiste de un *disparador* (trigger), *receive(Actualizar(Edo_Nodo,Edo_Disp),TD)*. En este caso un evento receive es transmitido a esta tarea. Los parámetros del evento disparador son el mensaje (*Actualizar(Edo_Nodo,Edo_Disp)*) y *TD* del cual el mensaje fue enviado. En esta transición se pueden apreciar dos actividades *Guardar(Edo_Nodo,Edo_Disp); Actualizar_Pasivos( )* las cuales se realizan cuando se recibe la transición.

Las dos transiciones salientes del estado Localizar_Conexión y la transición del estado de Funcionalidad contienen una condición guardia, estas tres transiciones finalizan al estado Activar, las condiciones guardias son excluyentes, por lo que al estado Activar llegará aquella transición cuya condición sea evaluada en verdadera. El Estado activar enviará la transición *^send(activar_disp(disp),TSP)* al rol correspondiente establecido por la transición que haya llegado a este estado; una vez finalizada pasa al estado Esperar_Activación, en el cual se espera por un evento *receive(activado,TSP)*, al recibir esta transición se ejecutará las acción de la transición *Actualizar_Pasivos( ).*

Estados

Los estados pueden contener actividades, las cuales son utilizadas para representar razonamiento interno, la percepción desde sensores, o realizar acciones por medio de actuadores (efectores). Actividades múltiples pueden estar incluidas en un solo estado y ser llevadas a cabo en secuencia. Una vez en el estado, la tarea permanece en este estado hasta que la actividad que se está realizando esté completa y una transición fuera del estado se vuelva activa. Las actividades están compuestas por funciones. Cada función puede regresar un resultado y puede tener un número de parámetros de entrada.

La sintaxis de un enunciado de actividad se muestra a continuación:

$$resultado = nombre\text{-}de\text{-}actividad(parametro_1, parametro_2, \ldots parametro_n)$$

Resultados múltiples pueden ser regresados de una actividad que usa notación tal como:

$$<x, y> = divide(a, b)$$

Una vez que un valor múltiple ha sido regresado en una tupla, las variables individuales que conforman la tupla pueden ser referidas como variables independientes. Las semánticas de las tareas concurrentes se basan en autómatas de estados finitos. Sin embargo, debido a que un solo rol esta compuesto por un determinado número de tareas concurrentes, el estado del rol esta compuesto por el conjunto de tareas concurrentes en cada una de las tareas concurrentes activas del rol.

Puesto que las actividades ocurren en los estados, los roles se encuentran generalmente en un estado por una cantidad finita de tiempo. Es importante señalar que en MaSE las transiciones entre los estados ocurren instantáneamente. Esto permite que el estado del agente sea determinado con precisión en cualquier momento en el tiempo.

Las variables usadas en las definiciones de las actividades, en estados, en mensajes y definiciones de eventos en las transiciones se asume que son visiblemente globales dentro de la tarea, pero no fuera de la tarea. Esto no significa que las variables usadas en la definición de una tarea son visibles dentro de las actividades definidas en la tarea. La única manera de transferir información desde una tarea a una actividad es por medio de parámetros pasados a la actividad o al resultado retornado de la actividad. La única forma de pasar información entre modelos de tarea es pasar información explícitamente como parámetros en una llamada de evento.

Todos los mensajes enviados entre agentes y eventos enviados entre tareas son puestos en una estructura de datos tipo cola. Esto permite asegurar que todos los menajes sean recibidos, aún si el agente o la tarea no están en el estado apropiado para manejar el mensaje o el evento inmediatamente. La tarea busca en la cola los mensajes que pueden ser manejados en el estado actual, aunque los mensajes y eventos del mismo tipo son manejados en el orden que son recibidos.

Cada tarea está exactamente en un estado en cualquier momento en el tiempo. Esto significa que las transiciones entre estados son instantáneas mientras que los estados toman tiempo. Generalmente, lo estados son usados para dos propósitos: esperar por un evento o realizar procesamiento interno. Si no hay actividades en un estado en particular o todas las actividades han sido completadas y no hay transiciones que hayan sido activadas, entonces la tarea está esperando sin hacer nada a que una transición sea activada o permitida.

Todas las actividades ocurren en un estado y son ejecutadas secuencialmente. Cuando una tarea incorpora un estado, la primera actividad es automáticamente ejecutada. Al terminarse la primera actividad, las actividades subsecuentes son ejecutadas una por una. Ninguna transición fuera del estado es activada hasta que todas las actividades hayan sido completadas. La meta de las tareas concurrentes es definir el comportamiento de los agentes, y al unir el razonamiento interno de los procesos del agente a su interacción con otros procesos internos así como también externos con otros agentes. Entonces se clasificaron por los desarrolladores de esta metodología a las tareas en tres tipos: reactivas, proactivas, o heterogéneas.

Una *tarea reactiva* tiene por lo menos un estado *idle* (inactivo) donde espera por una petición de otra tarea o bien, de un agente antes de empezar algún procesamiento. Una *tarea reactiva* siempre empieza en uno de sus estados *idle* (inactivos). Esto es, el estado *start* tiene una transición *null* (nula) a su estado *idle.*

Las *tareas proactivas* no tienen estados idle (inactivos). Estas tareas están continuamente generando peticiones para otros agentes o tareas.

Una tarea heterogénea, es una combinación de las tareas reactivas y proactivas. Una tarea heterogénea tiene estados *idle,* pero no empieza en un estado *idle (inactivo).* Genera por lo menos una petición para otro agente o tarea antes de entrar en un estado idle (inactivo). Basándonos en estas definiciones de tareas, entonces es posible categorizar agentes cuyos comportamientos estén compuestos por tareas ya sea proactiva o reactiva.

Un agente proactivo es un agente con por lo menos una tarea proactiva o heterogénea, mientras que un agente reactivo es un agente con todas las tareas reactivas, los tareas que se definieron para nuestros sistema tolerante a fallos son de tipo reactivas.

La estructura de todas las tareas de las que consiste el SMA tolerante a fallos par el SCDRM se pueden apreciar en el CD anexo.

Hasta aquí se concluyó la fase de análisis de nuestro Sistema Multiagente: Ya que concluyó la fase de captura de metas al estructurar en el Diagrama Jerárquico de Meta. Además, se desarrollaron una colección de casos de meta que se emplearon para definir los diagramas de secuencia, así mismo se desarrollaron cada una de las tareas concurrentes que constituyen los roles que integran nuestro sistema. Por lo que siguiendo los pasos de la metodología continuaremos con la fase del diseño donde introduciremos el proceso de análisis como soporte del diseño.

6.3 Modelando el Sistema Multiagente en su fase de diseño

6.3.1 Creación de las clases de agentes

En la fase de diseño el primer paso fue crear las clases de agentes que integran a nuestro Sistema Multiagente, esto lo realizamos a partir de los roles. El producto que obtuvimos de esta fase, fue el *Diagrama de Clases de Agente,* la cual muestra las clases y las conversaciones entre ellos. Este diagrama es el primer diseño del agente en MaSE que muestra el Sistema Multiagente completo.

Una clase es un plantilla o patrón para un tipo particular de agente, el cual estará en el sistema. En esta fase, las clases tienen dos componentes, roles y conversaciones. En una etapa posterior realizamos los detalles internos que fueron agregados a nuestras clases.

Cada rol generalmente debe estar contenido en una clase, en está metodología un rol se puede desempeñar por dos clases de agentes o una clase puede representar varios roles y cambiar dinámicamente.

Conectamos al las diferentes clases de agente mediante las conversaciones. De hecho, el propósito principal de esta fase es la de identificar las clases de agente que participan en cada lado de una conversación entre dos agentes diferentes.

Cuando las clases de agentes heredan trayectorias de comunicación entre roles, forman conversaciones con otras clases de agentes. Son estas conversaciones la estructura principal de

un MAS, ya que logran realmente la distribución del sistema, siendo esto la fortaleza de la tecnología de agentes.

En el diagrama de clases de agente que se genera, cada rectángulo representa una clase agente y cada flecha representa una conversación entre clases de agente, la dirección de las flecha indica al iniciador y al respondedor de la conversación.

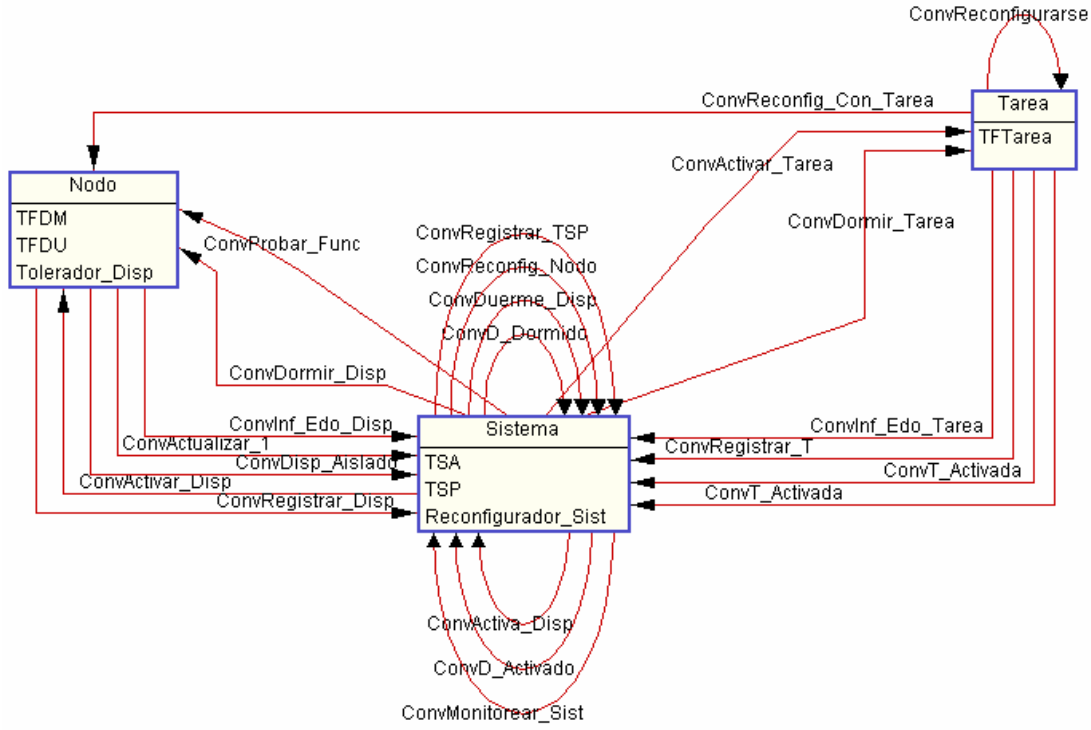


Figura 6. 11 Diagrama de clases de agente para el SMA que tolera los fallos en el SCDRM.

La figura 6.11 muestra el diagrama de clases de agente para nuestro sistema, en él se observan tres clases de agentes:

1. Nodo: encargado del monitoreo de los dispositivos únicos y múltiples, esta clase agente desempeña tres roles (TFDM, TFDU y Tolerador_Disp).
2. Sistema: encargado del monitoreo del Sistema Activo, de los Sistemas Pasivos y de reconfigurar al SCDRM al fallar un dispositivo o un nodo en el sistema. Esta clase agente contiene tres roles (TSA,TSP y Reconfigurar_Sist).
3. Tarea: a cargo del monitoreo de las tareas existentes en el SCDRM, contiene el rol TFTarea.

El hecho de que una clase agente contenga más de un rol no significa que los jugará en el mismo momento. El rol que desempeña una clase agente puede ser cambiado en tiempo de ejecución de acuerdo a las acciones que sucedan en el sistema. En nuestro caso, por ejemplo, la *clase agente Nodo* inicia con el rol Tolerador_Disp ya que la primer tarea que debe realizar el agente es registrarse con el TSA, posteriormente jugará el rol TFDU si ese agente esta a cargo de un dispositivo único.

Los roles TSA y TSP no pueden ser desempeñados al mismo tiempo por la clase agente, ya que de acuerdo a los requisitos planteados en el capítulo 4, un nodo contendrá solamente un agente sistema activo o pasivo, aunque en algún momento el rol TSP puede desempeñar el rol TSA al fallar el nodo en el que éste último se encuentra.

En la figura 6.11 se puede observar las conversaciones existentes entre las diferentes clases de agentes, estas conversaciones contienen la interacción que existe entre las tareas del modelo de roles.

Hasta este punto se han definido las clases de agentes y asignado los roles que desempeña, las conversaciones solo han sido establecidas, más no desarrolladas. La herramienta AgentTool que emplea MaSE contiene integrado una técnica formal que nos permite obtener las conversaciones de las tareas, el diagrama de clase de agente resultante puede ser modificado. En la siguiente sección se describe a detalle como se obtienen las conversaciones que conforman el diagrama de clases de agentes mediante una serie de pasos llamado proceso de transformación.

6.3.2 Construyendo conversaciones en SMA que tolera los fallos en el SCDRM

Los agentes dentro de un Sistema Multiagente logran comunicarse mediante mensajes estructurados. Por lo que al diseñar un SMA se debimos estar seguros que las tareas que diseñamos cumplan con las metas que establecimos en la fase de análisis.

Hay ocasiones en la que los agentes se utilizan en los ambientes abiertos donde se pueden encontrar con cualquier tipo de contingencia. En un ambiente abierto, un agente debe poder construir dinámicamente sus conversaciones. Sin embargo, esta metodología solo se refiere a los ambientes cerrados donde los agentes están enterados del entorno que los rodea y saben quiénes son los demás agentes que integran el SMA. Siempre que un agente envía o recibe un mensaje, éste pasa a través de varios estados en una conversación, estos estados determinan cómo el agente se debe de comportar.

Específicamente, una conversación consiste de dos diagramas de comunicación, un diagrama para la clase que inicia la conversación y el otro diagrama para la clase que responde a la citada conversación. Cada diagrama que interviene en la conversación es representado por medio de un autómata de estados finito que define los estados de la conversación de las dos clases que intervienen.

La figura 6.12 y 6.13 muestran los diagramas de comunicación para la conversación *ConRegistrar_Disp*, esta conversación es realizada por la clase agente Nodo (iniciador) y la clase agente Sistema (respondedor). La clase Nodo inicia la conversación enviando el mensaje *registrar(parent.TFNTS, parent.Tipo)* a la clase agente Sistema, este lo recibe, registra al agente nodo, se actualiza y actualiza a los demás agentes sistemas, al finalizar le envía el identificador al agente nodo mediante el mensaje *registrado(regID)*, el agente nodo lo recibe y se actualiza.

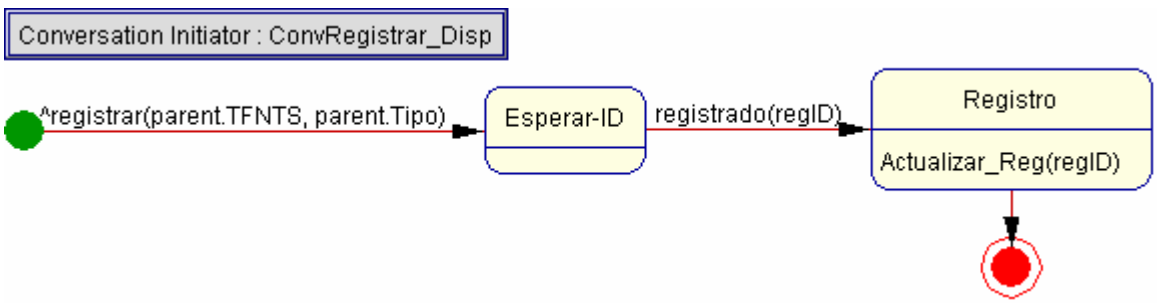


Figura 6. 12 Iniciador de la mitad de la conversación ConRegistrar_Disp.

Figura 6. 13 Respondedor de la mitad de la conversación ConRegistrar_Dis.

Cuando un agente recibe un mensaje, lo compara con sus conversaciones activas. Si hay una coincidencia, el agente mueve la conversación apropiada a un nuevo estado y realiza cualquier acción requerida ya sea de la transición de un nuevo estado. De otra manera, el agente compara el mensaje con todas las posibles conversaciones en las que pueda participar con el agente que mandó el mensaje, y comienza una nueva conversación si el mensaje concuerda con una transición del estado de inicio.

El problema principal se encuentra en conocer que estados y transiciones agregar al diagrama de clases de comunicación. Los diagramas de secuencia y las tareas fueron creados para asegurar que cada mensaje o evento capturado pueda ser traducido en conversaciones, tales como los eventos *send* y *receive*.

Las conversaciones deben ser consistentes con todos los diagramas de secuencia diseñados con anterioridad. También deben incorporar los estados de las tareas. Algunas tareas de hecho, operan totalmente sobre una conversación sencilla y pueden ser diseñadas directamente. En general, las conversaciones son construidas al agregar primeramente todos los posibles estados y transiciones que pueden ser desprendidos del diagrama de secuencia y del diagrama de tareas concurrentes.

AgentTool tiene integrada una herramienta que nos permite obtener directamente los diagramas de las conversaciones, estas conversaciones pueden ser eliminadas o modificadas. En las siguientes secciones se describe a detalle como se obtienen los diagramas de las conversaciones a partir de las tareas de los roles por medio del proceso de transformación.

Mientras se diseñan las conversaciones, es indispensable que sean verificadas durante el diseño para evitar que colapsen. En general, una conversación se encuentra en un punto muerto cuando ambos lados de la conversación están esperando un determinado mensaje. Existen dos maneras de evitar que esto suceda, para cada *send* de un lado, debe de existir un *receive* del otro lado, y además, cada conversación debe poder salir de cada estado, en otras palabras, esto es que cada estado debe tener una transición válida que lo lleve al estado final de la conversación en el diagrama, las tablas que diseñamos (Anexo C)nos sirvieron de mucha ayuda para realizar esta etapa de nuestro diseño.

Los ciclos infinitos, deadlock (callejón sin salida) y otros errores en la comunicación pueden ocasionar trastornos cuando se diseña un SMA, peor aún, el sistema puede seguir trabajando mientras que existe un problema que es catastrófico y desapercibido cundo se realiza el diseño de la conversación. Por lo que es necesario explorar las trayectorias para que la conversación pueda ser factible y válida, para esto se requiere verificarla formalmente. Una vez que se hayan verificado las conversaciones, se puede confiar que los agentes se comunicarán según lo esperado.

La herramienta AgentTool provee un módulo que asegura la validez e interoperabilidad de las conversaciones, también en éste módulo logra que se pueda cumplir la política del protocolo de comunicación en un determinado sistema propuesto.

Por lo que MaSE a través de su herramienta de desarrollo diseñó el módulo que utiliza una técnica formal para verificar automáticamente que los protocolos de comunicación definidos en el ambiente de un SMA y estos sean completamente válidos. Esto se puede lograr examinando las conversaciones del agente antes de desplegar el sistema. La técnica formal de validación implementada para MaSE se explica de manera general en las siguientes secciones, para una descripción más detallada consulte [Lacey 2000].

6.3.3 Ensamblando clases de agentes en SCDRM

En esta fase de MaSE, las partes internas de un agente son creadas. Robinson [Robinson 2000] describe los detalles del ensamblaje de los agentes desde una base arquitectónica de componentes. Esta sección provee una breve descripción del proceso.

Las herramientas para describir las clases individuales de los agentes son presentadas por Robinson. Él provee cinco diferentes plantillas de arquitecturas y describe el acercamiento orientado a objetos creado por él. Las arquitecturas incluidas son: BDI, reactiva, planeación, basada en conocimiento y agentes definidos por el usuario. Cada plantilla de arquitectura tiene un número específico de componentes. Por ejemplo, una arquitectura reactiva incluye un Controller, MessageInterface, RuleContainer, y Effectors.

Por lo que se puede definir los componentes a utilizar o bien se puede utilizar una serie de componentes ya predefinidos. Además, los componentes pueden contener sub-componentes que bien puede ser parte de la arquitectura. En teoría los sub-componentes pueden continuar indefinidamente, auque en la práctica pueden estar raramente sub-definidos. Los componentes son declarados instantáneamente para producir objetos reales del código. Las instancias dependen de los componentes, pero requiere ya sea de la selección de objetos precodificados o de la generación de código.

Los componentes se conectan ya se con conectores internos o externos. Los conectores internos son conectados con otros componentes y los conectores externos son conectados con recursos externos como otros agentes, sensores y actuadores, bases de datos. Además, los componentes son asociados con los diagramas de estado que representan una secuencia de eventos que son pasados de un componente a otro. Un ejemplo es mostrado en la figura 6.14.

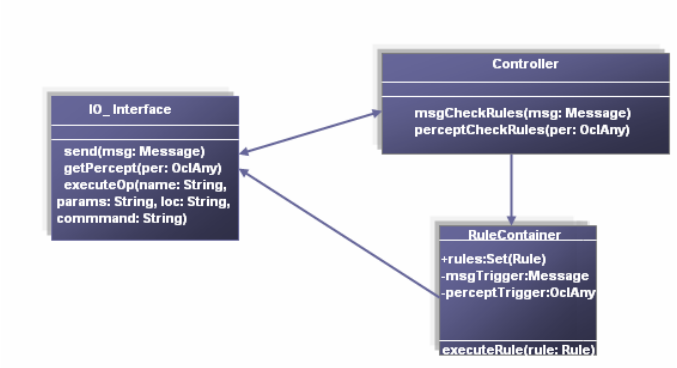


Figura 6.14 Arquitectura Reactiva del un agente

6.3.4 Diagrama de despliegue del SMA que tolera los fallos en el SCDRM

La fase final MaSE, toma las clases de agentes y las dirige como verdaderos agentes. Usa un diagrama llamado *Diagrama de Despliegue* para mostrar los números, tipos, y localizaciones de los agentes dentro del sistema. Esta es la fase más sencilla, ya que la mayor parte del trabajo fue realizada en los pasos anteriores.

El sistema se puede analizar en el *Diagrama de Despliegue* antes de que pueda ser implementado en código. Esto se debe a las diferencias que existen entre los agentes y las clases del agente. Un agente requiere de información tal como el hostname y una dirección para participar en cualquier comunicación externa al sistema en el cual resida

El *Diagrama de Despliegue* también nos ofrece otras oportunidades para ajustar el sistema. Los agentes pueden ser ordenados por varias configuraciones de máquinas, para utilizar de la mejor manera la fuerza del procesamiento o ancho de banda.

En algunos casos, los requerimientos de sistema pueden especificar un número particular de componentes, o de máquinas particulares en las que residen. De otra manera, se debe considerar el tráfico de mensajes al momento de colocar agentes en máquinas particulares. Obviamente, la velocidad de comunicación entre agentes dependerá de la red en la que se comunican. En varios casos, los agentes pueden ser puestos en la misma máquina. Al poner varios agentes en un solo equipo se destruye las ventajas de la distribución obtenidas al usar el paradigma de agentes.

Otra consideración es el poder de procesamiento en un equipo particular y el requerido por un agente en específico. Si un agente tiene un alto requerimiento de CPU, puede ser posicionado en una máquina para él únicamente. Una fortaleza de MaSE es que estas modificaciones pueden ser hechas después de diseñar y de generar una variedad de configuraciones de sistema, junto con la reunión de los datos del funcionamiento.

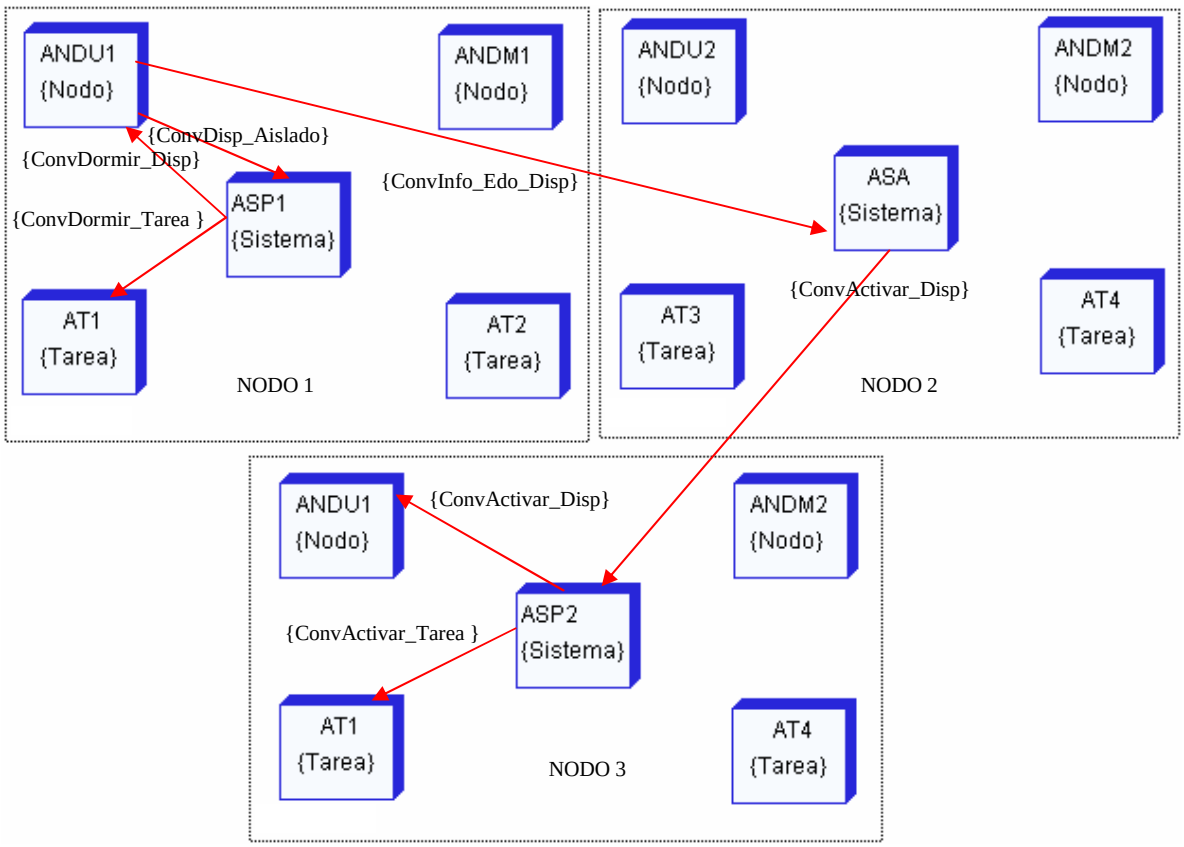


Figura 6. 15 Diagrama de Despliegue del SMA que tolera los fallos en el SCDRM.

La figura 6.15 muestra el *Diagrama de Despliegue* para el SCDRM (puede tener múltiples configuraciones), se muestran 3 nodos; el nodo 1 contiene al agente nodo dispositivo único 1 (ANDU1) y al agente tarea 1 (AT1) que deben trabajar con el dispositivo único, además contiene al agente nodo dispositivo múltiple 1 (ANDM1) y agente tarea 2 (AT2) que trabajan con un dispositivo múltiple, y por último contiene al agente sistema pasivo 1 (ASP1) a cargo de ese nodo; el nodo 2 contiene al agente nodo dispositivo único 1 (ANDU2), el agente tarea 3 (AT3), agente nodo dispositivo múltiple 2 (ANDM2), agente tarea 4 (AT4) y el agente sistema activo (ASA); el nodo 3 contiene al agente nodo dispositivo único (ANDU1) inactivo a cargo del dispositivo único (replica del contenido en el nodo 1) y su correspondiente agente tarea 1 (AT1) copia del (AT1) contenido en el nodo 1, de igual manera se encuentran de manera replicado el ANDM2 y AT4 correspondientes al ANDM2 y AT4 del nodo 2, por último contiene al ASP2 a cargo de ese nodo. En la figura se muestran las conversaciones requeridas para cuando falla el dispositivo único del nodo 1 (monitoreado por el ANDU1). En éste diagrama se deben incluir todas las conversaciones del diagrama de clase de agentes, para efecto de mejor visibilidad se muestran solo algunas conversaciones.

Las cajas tridimensionales son agentes y las líneas que conectan representan conversaciones entre los agentes. Cualquier conversación entre las clases del agente aparece entre los agentes de esas clases. Además, una caja de línea discontinua indica que los agentes están contenidos en la misma plataforma física.

6.4 Transformación de la fase de análisis a la fase de diseño

El proceso de transformación proporciona un método correcto y robusto para generar los modelos del diseño de MaSE partiendo de los modelos del análisis sin perder ninguna información a partir de la fase de análisis. Los sistemas formales de transformación reducen las equivocaciones incurridas durante el diseño.

El sistema de transformación fue un proceso automático, requiriendo solamente algunas decisiones del diseño de nuestra parte. Las ventajas al contar con este proceso automatizado es que nos permitió preservar la corrección a partir de un modelo al siguiente, ya que nos proporcionó una traceabilidad clara entre el análisis y el diseño, nos simplificó posteriormente el proceso de verificación de conversaciones, además nos permitió tener más confianza en que ninguna inconsistencia o error ocurrieron durante el proceso del diseño.

AgentTool tiene incluido un modelo de transformación que nos permitió pasar de la fase de análisis a la fase de diseño. La transformación de la fase de análisis a la fase de diseño en MaSE se puede definir como una serie de pasos pequeños que manipulan el modelo en una representación alternativa. Cada transformación la pudimos realizar de manera determinista en su ejecución y no se introdujeron inconsistencias entre los modelos. Además AgentTool, nos permitió regenerar el diseño cuando realizamos cambios en nuestro sistema.

Esta sección describimos la transformación realizada en nuestra propuesta, donde se toman los modelos del análisis y se producen los correspondientes modelos de diseño dentro del contexto de la metodología de MaSE. Para obtener una mayor descripción de éste proceso consultar [Sparkman 2001].

6.4.1 Punto de partida del proceso de transformación

Antes de realizar las transformaciones nos aseguramos de haber desarrollado los siguientes modelos en la fase de análisis:

- El Modelo de Roles.
- Los Diagramas de Tareas Concurrentes de las tareas que contiene cada rol especificado en el modelo de roles.
- Haber Definido el conjunto inicial de la clase de agentes, pero solamente al grado de decidir cuál conjunto de roles desempeñará cada clase agente, asegurándonos de que cada rol es desempeñado por lo menos en una clase de agente.

En la metodología MaSE se tiene claro que los roles que una clase de agente desempeña, conjuntamente con las trayectorias de comunicación entre las tareas de los roles, determinan las conversaciones que cada clase de agente tendrá.

Es importante tomar en cuenta que, cuando dos roles se combinan en una sola clase de agente, se debe determinar si los protocolos entre los roles deben permanecer como comunicación externa o si la comunicación que el protocolo representa ahora es una comunicación interna dentro de la clase del agente, puesto que los protocolos externos representan mensajes que pasarán entre los agentes, y se convertirán en una o más conversaciones. La razón que puede no ser una sola conversación es porque la comunicación entre los agentes puede no ser continua. Podría haber otra comunicación coordinada que debe ocurrir internamente, o con otros agentes. Sin embargo, los protocolos internos (inicialmente definidos o cambiados cuando los roles son combinados) no darán lugar a conversaciones que representen esa comunicación. En segundo

lugar, si más de una clase de agente desempeña un rol, entonces para cada protocolo externo que implique ese rol, las conversaciones serán creadas para cada clase de agente que desempeña ese rol. Esto significa que habrá múltiples instancias de la misma conversación entre diversos agentes del conjunto.

En la figura 6.16 se puede visualizar como se realiza la transformación de la fase de análisis a partir del modelo de roles y las tareas concurrentes a la fase de diseño en los modelo de la clase de agentes, sus componentes internos y las conversaciones. La figura 6.16 indica que por cada tarea que exista en los roles dentro del modelo de roles se creará un componente, además se crearán las conversaciones requeridas a partir de cada componente, este proceso se describe en los pasos que se siguen al realizar la transformación.

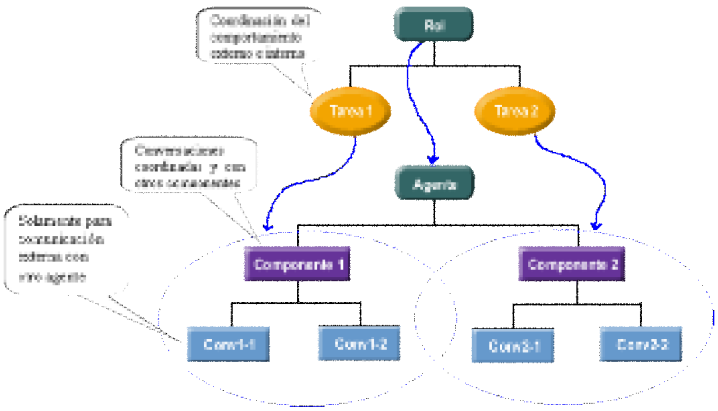


Figura 6. 16 Modelo de transformación: componentes de la fase de análisis a componentes de la fase de diseño.

Los diseñadores de MaSE aseguraron que la única manera de modelar la organización de la estructura de los agentes, componentes, y de conversaciones en la fase del diseño, es capturando toda la información que está presente en los modelos del análisis y conservar la idea básica de una conversación. Ahora bien, se puede decir que las conversaciones que se recolectan de las tareas pueden ser pequeñas partes que forman la comunicación total que ocurre entre dos agentes. La razón por la que la comunicación es separada en múltiples pedacitos es porque hay otra comunicación sin relación, ya sea con eventos internos o comunicación con otro agente, que debe ocurrir en los diversos pedacitos.

Un sistema de transformación fue implementado como parte del ambiente del desarrollo de AgentTool. La figura 6.17 muestra el menú que fue agregado a la barra de menú de AgentTool. El item del menú *Add Agent Components* corresponde a la primera etapa del proceso de transformación, *Annotate Component State Diagrams* corresponde a la segunda etapa, y *Create Conversations* corresponde a la tercera etapa. Se insiste antes de que las transformaciones puedan ocurrir el modelo de roles debe existir y debe haber por lo menos una clase del agente que desempeña cada rol.

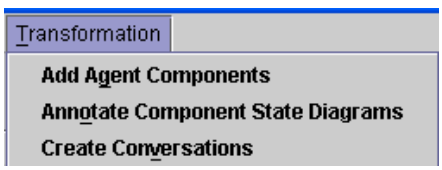


Figura 6.17 Menú para poder realizar el proceso de transformación con AgentTool.

En esta sección se describe como se llevó a cabo la transformación del análisis al diseño de nuestro sistema. En la figura 6.18 se muestra el proceso de transformación a partir del modelo de roles, en esta figura solo se observan los roles, tareas y protocolos que intervienen para llevar a cabo lo descrito en el caso de meta denominado Tolerador de Fallos a Nivel Nodo en un Dispositivo Único. No SE demuestra cada situación posible que pueda presentarse, para más información sobre el proceso de transformación consultar [Sparkman 2001].

El modelo de roles para el SMA tolerante a fallos en el SCDRM consta de cinco roles, con sus respectivas tareas. El rol TDFU es responsable de monitorear a los dispositivos únicos encontrados en los diferentes nodos que integran el SCDRM, para ello es necesario desempeñar ciertas tareas para lograr sus objetivos definidos por las metas que lo integran. El rol TSA y TSP se encargan de reconfigurar a los dispositivos y tareas al momento de ocurrir un fallo, además el rol TSA registra a los agentes nodos. Los roles TFTarea y Tolerador_Disp tienen la función de detectar, aislar y recuperar de un fallo a la tarea y al dispositivo respectivamente y reconfigurar a las tareas o dispositivos en determinadas ocasiones de acuerdo a las acciones enviadas por los roles TSA y TSP; asimismo el rol Tolerador_Disp tiene la responsabilidad de registrar a su agente con el rol TSA. La interacción entre las diferentes tareas que constituyen al modelo de roles se da mediante los protocolos mostrados en la figura 6.18.

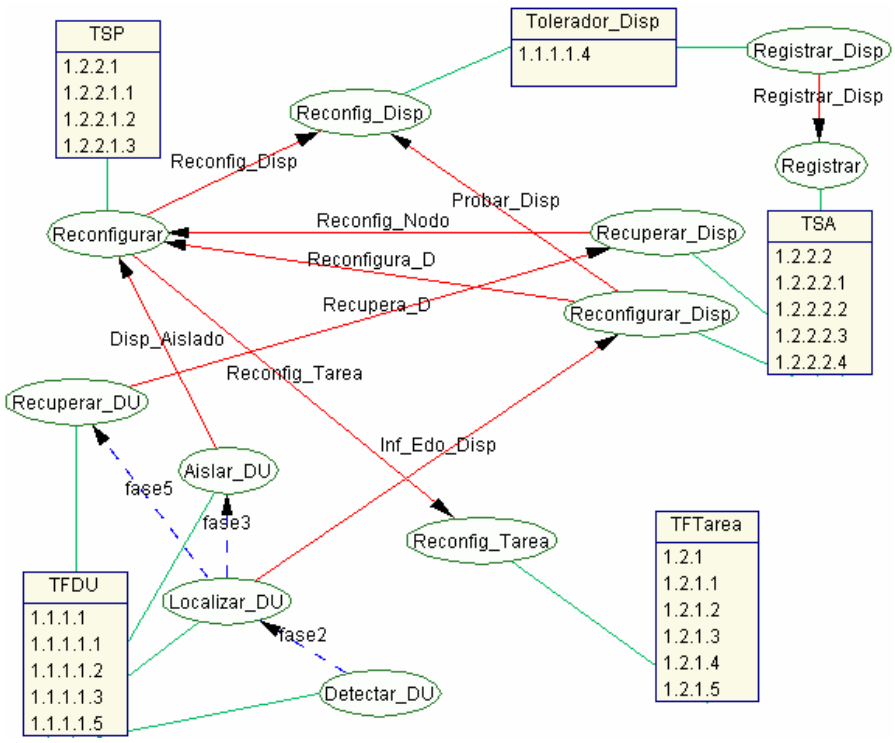


Figura 6.18 Modelo de roles para el SMA tolerante a fallos en SCDRM.

La figura 6.19 muestra las clases de agentes para el modelo de roles presentados en la figura 6.18. Los roles TFDM y *Reconfigurador_Sist* no se observan en la figura 6.18 pero son requeridos para el sistema global, ya que en el proceso de transformación requiere que sean incluidos todos los roles existentes para llevarse a cabo.

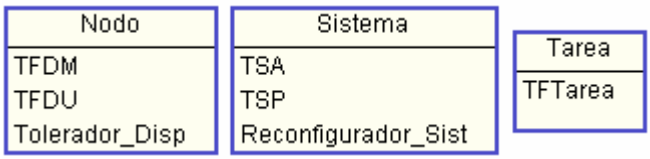


Figura 6.19 Clases de agentes en el SMA tolerante a fallos en SCDRM.

Para mostrar el seguimiento del proceso de transformación a través de sus pasos solo se tomarán como ejemplo algunas tareas del modelo de roles de la figura 6.18. A continuación se realiza una breve descripción de dichas tareas.

La figura 6.20 muestra el diagrama de estado que representa la tarea *Localizar* del rol *TFDU*. Esta tarea se encuentra en un estado inactivo hasta que recibe un mensaje interno (*Localizar*) mediante el protocolo (interno) *fase 2*, el cual indica que ha ocurrido algún fallo en el dispositivo. La tarea corrobora el fallo del dispositivo, si no hay fallo se pone como reestablecido al dispositivo, si existe fallo se le notifica de ello al rol *TSA* mediante el protocolo externo *Inf_Edo_Disp* y envía dos eventos internos: *Aislar* y *Recuperar* por medio de los protocolos internos *fase 3* y *fase 5* respectivamente.

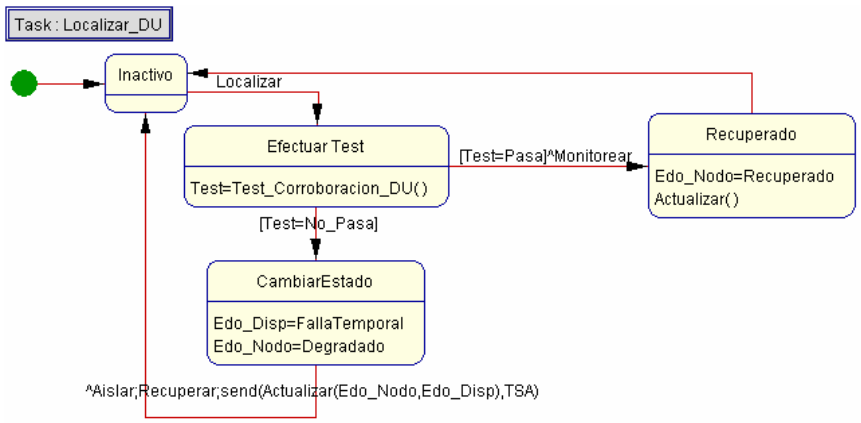


Figura 6.20 Tarea Localizar_DU del rol TFDU.

La figura 6.21 define la tarea *Reconfigurar_Disp* del rol *TSA*. La tarea se mantiene inactiva hasta el momento de recibir un mensaje *receive(Actualizar(Edo_Nodo,Edo_Disp),TD)* del rol *TFDU*, en ese momento busca el nodo donde se encuentra la replica del dispositivo, le envía un mensaje a la tarea *Reconfig_Disp* del rol *Tolerador_Disp* por medio del protocolo *Probar_Disp* para que verifique la funcionalidad del dispositivo y si existe replica le informa al rol *TSP* que active al dispositivo y tareas en su nodo por medio de la tarea *Reconfigurar*, la cual emplea el protocolo *Reconfigura_D*, Debe de activar al dispositivo en la replica.

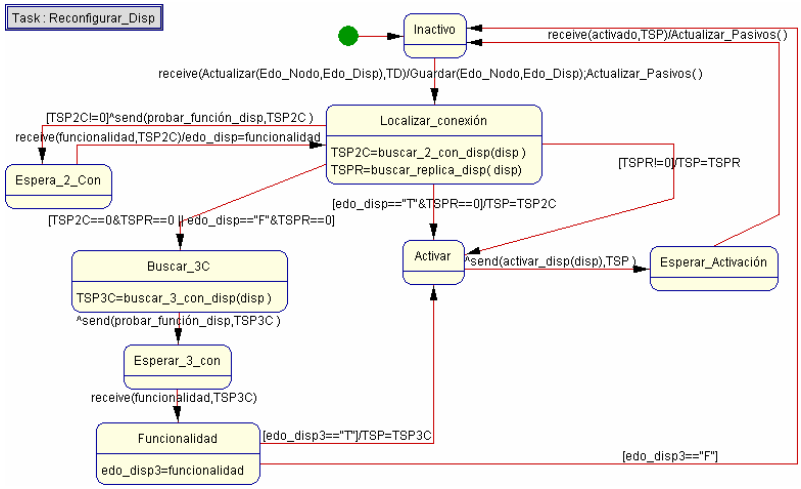


Figura 6.21 Tarea Reconfigurar_Disp del rol TSA.

La figura 6.22 muestra la tarea *Reconfigurar* del rol *TSP*, esta tarea se encarga de reconfigurar a los dispositivos de su nodo. Por ejemplo, cuando el *TFDU* detecta fallo aísla al dispositivo y se lo informa al rol TSP, esa información es recibida en esta tarea mediante el mensaje *receive(disp_aislado(disp),TD)* el cual pertenece al protocolo *Disp_Aislado*, al recibir este evento el *TSP* le informa al *TFDU* del dispositivo que falló que desactive a dicho dispositivo, el *TDFDU* al desactivar el dispositivo le informa al TSP, éste al recibir el mensaje le informa al *TSA* que el dispositivo con fallo ha sido desactivado y les informa a los TFTareas de las tareas dependientes al dispositivo que desactiven su respectiva tarea, la transición *receive(disp_dormido,TD) ^send(dormido,TSA); send(dormir_tarea,<tareas>)* muestra estas acciones. El mensaje *receive(activar_disp(disp),TSA)* recibido del estado Inactivo al estado *Activar* es enviado a la tarea por el rol *TSA* indicándole que debe activar al dispositivo. Los mensajes recibidos del estado inactivo pueden ser recibidos en diferentes tiempos, por ejemplo, el *TFDU* que detectó fallo en su dispositivo recibirá el mensaje de que desactive su dispositivo; el *TFDU* del nodo que contenga la replica de dicho dispositivo recibirá el mensaje de que active al dispositivo.

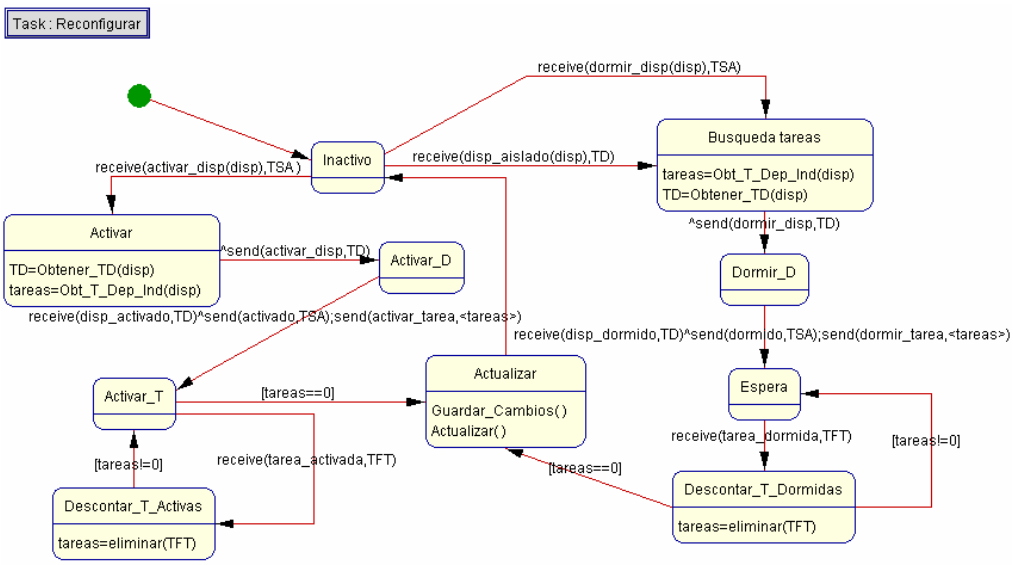


Figura 6.22 Tarea Reconfigurar del rol TSP.

La tarea *Reconfig_Disp* del rol *Tolerador_Disp* es representada por la figura 6.23, esta tarea reconfigura al dispositivo del cual esta a cargo, esta reconfiguración la realiza activando, durmiendo o probando la funcionalidad del dispositivo de acuerdo a las acciones tomadas por el TSP y el TSA.

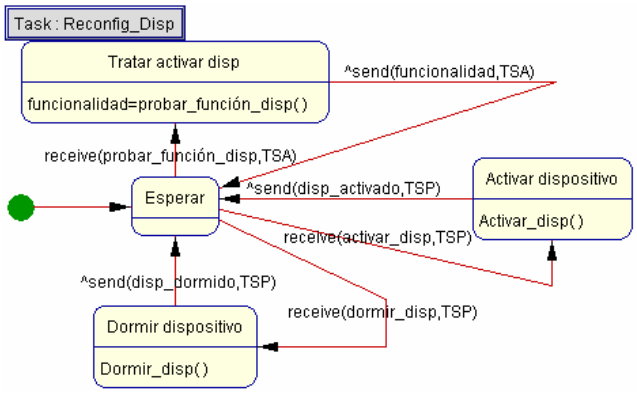


Figura 6. 23 Tarea Reconfig_Disp del rol Tolerador_Disp.

6.4.2 Primera etapa del proceso de transformación: Add Agent Components

Ya que definimos el modelo de roles y el diagrama inicial de clase del agente, la primera etapa del proceso de transformación la pudimos iniciar. En esta etapa, las transformaciones determinamos primero los conjuntos de protocolos que le pertenecen a cada evento externo. Y posteriormente determinamos los componentes para la clase de agentes se crearon basándonos en los roles ya asignados.

Para transformar eventos externos en eventos internos se requiere declarar los protocolos como interno, y los protocolos a los cuales cada evento externo pertenece primero se debe de determinar. Los eventos, o los mensajes que representan, pueden pertenecer a múltiples protocolos.

6.4.2.1 Determinando los protocolos para los eventos externos

La primera transformación en esta etapa se consiguió identificando a los protocolos para los eventos externos. En la mayoría de los casos, este proceso no requirió de ninguna entrada por parte de nosotros. Sin embargo, en algunos casos es imposible determinar automáticamente en qué protocolos los eventos fueron realizados. La pantalla de diálogo en la figura 6.24 se exhibe como información sobre lo que sucederá después.

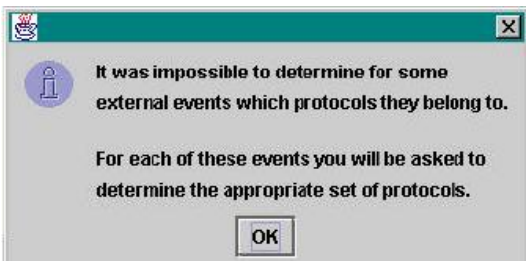


Figura 6.24 Dialogo de ambigüedades en protocolos.

Dado el trabajo realizado hasta éste momento no contiene ambigüedades se muestra el diagrama de roles de la figura 6.25 y también se muestran dos protocolos salientes de la tarea *Reconfigurar_Disp* del rol *TSA*, estos protocolos son *Probar_DispU* y *Probar_DispM*. Las tranformaciones no se pudieron determinar automáticamente los protocolos para el evento *send(probar_función_disp,TSP2C)* de la tarea *Reconfigurar_Disp*, ya que existen dos tareas que reciben éste mensaje. Por lo que se requirió tomar la decisión para los eventos. Según lo mostrado en el figura 6.26, la transición con el evento en la pregunta se destaca en la tarea *Reconfigurar_Disp* y otra ventana se exhibe para seleccionar los protocolos para éste evento.

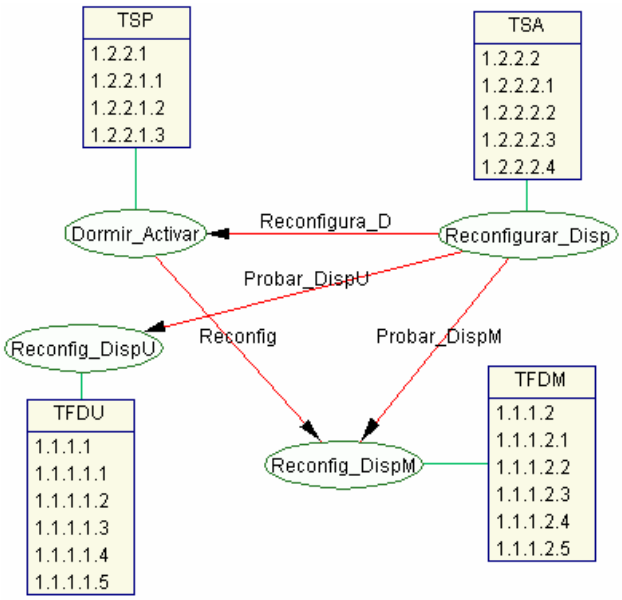


Figura 6.25 Caso en el que existen ambigüedades.

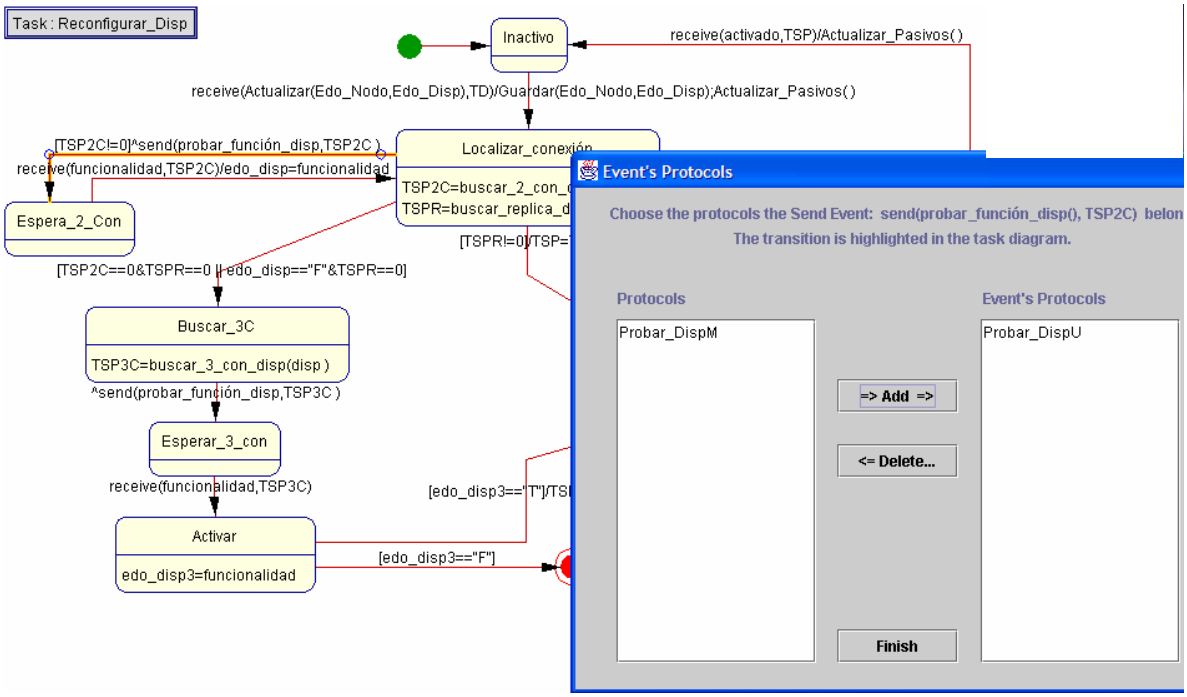
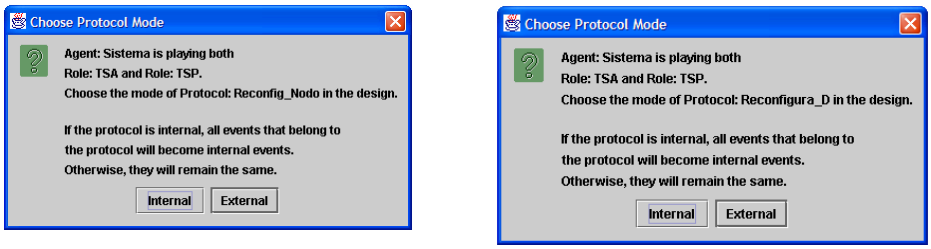


Figura 6.26 Decidiendo protocolos.

6.4.2.2 Determinando el modo para los protocolos

Como se muestra en la figura 6.19 se decidió que la clase agente Sistema podría desempeñar tres roles: TSA, TSP y *Reconfigurador_Sist*, en base a esto el proceso siguiente de las transformaciones es decidir sí los protocolos entre las tareas de esos roles seguirán siendo externos, o si serán una comunicación interna. La figura 6.27 muestra los cuadros de diálogos desplegados por el sistema de transformación para nuestra clase agente. Donde indica que la clase agente Sistema desempeña los roles TSA y TSP y así podemos decidir si los protocolos *Reconfigura_D* y *Recupera_Nodo* serán internos o externos dado que estos protocolos relacionan una tarea del rol TSA con una tarea del rol TSP. Para el primer protocolo es la tarea *Reconfigurar_Disp* y para la segunda es la *tarea Recuperar_Disp*, ambas tareas se comunican con la tarea Reconfigurar del rol TSP, dado que ambos roles están en la misma clase fue necesario decidir que tipo de evento son. Si seleccionamos el botón Internal cada evento que pertenece al protocolo de *Reconfigura_D* o *Recupera_Nodo* en los componentes de Reconfigurar y *Reconfigurar_D* se cambiarán en eventos internos. En nuestro caso ambos protocolos siguen siendo externos.



(a) ***(b)***

Figura 6.27 Cambiando protocolos.

6.4.2.3 Componentes del agente

El resultado de la primera etapa del proceso de transformación es que los componentes son definidos para las clases de agente basadas en los roles que desempeñan. La clase agente contendrá un componente por cada tarea contenida en los roles que desempeña ésta. El diagrama de estado para el componente es inicialmente igual que el de la tarea implementada. Si hay algunos eventos externos que pertenezcan a los protocolos y determine que son comunicación interna, los eventos se transforman en eventos internos (la descripción *send* y *receive* son removidos de la transición de las tareas). Los componentes de las clases de agente para nuestro sistema quedaron igual que el de la tarea, ya que el tipo de protocolo no cambió. Al término de la primera etapa del proceso de transformación se obtuvieron los componentes internos de la arquitectura de los agentes. La figura 6.28 muestra esta arquitectura para la clase Nodo, para la clase Sistema se observa en la figura 6.29 y la figura 6.30 muestra la arquitectura de la clase Tarea. Esta arquitectura se obtuvo del modelo de roles de la figura 6.18. Cada componente internamente contiene hasta esta transformación el diagrama del componente, que en nuestro caso es igual al diagrama de la tarea concurrente de la cual se derivó.

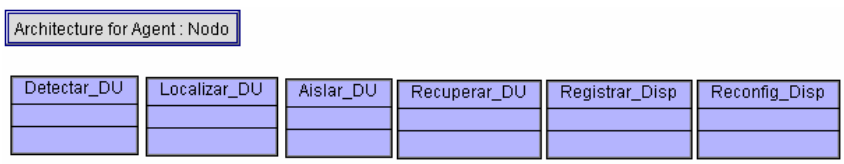


Figura 6.28 Arquitectura interna del agente Nodo al término de la primera etapa del proceso de transformación.

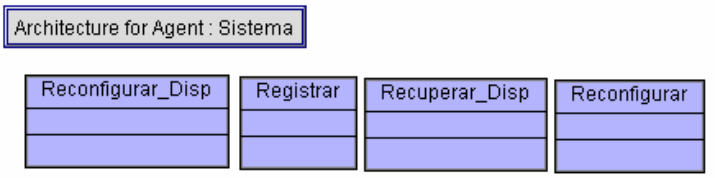


Figura 6. 29 Arquitectura interna del agente Sistema al término de la primera etapa del proceso de transformación.

Figura 6.30 Arquitectura interna del agente Tarea al término de la primera etapa del proceso de transformación.

6.4.3 Segunda etapa del proceso de transformación: Annotating Component State Diagrams

La segunda etapa se centra en la anotación de los componentes del diagrama de estado para mostrar dónde las conversaciones inician y dónde terminan, esta etapa también concuerda con los eventos externos en los diferentes componentes que se convierten en los mensajes iniciales de una conversación.

6.4.3.1 Concordando los primeros mensajes de las conversaciones

La primera interacción de la etapa dos requirió determinar analizar si correspondían los eventos en diferentes componentes. En la mayoría de los casos esto se puede hacer automáticamente, pero como con la determinación de los protocolos para los eventos, hay algunos casos donde solamente nosotros pudimos tomar la determinación. En estos casos, una ventana se despliega con los diagramas de estado de ambos componentes que contienen los eventos en cuestión, también en el modelo de roles a partir de la fase de análisis para la referencia. Las transiciones en los diagramas de estado que contienen los eventos se destacan, y se requiere cuestionar si corresponden los eventos el uno con el otro.

En nuestro sistema, hay cuatro casos donde las transformaciones no pueden determinar automáticamente los eventos correspondientes. El primer caso, es mostrado en las figuras 6.31 y 6.32, implican los mensajes tarea desactivada o dormida y tarea activada respectivamente, perteneciendo al componente *Reconfig_Tarea* del agente Tarea, y al componente Reconfigurar del Agente Sistema. Estos eventos fueron pensados para corresponder, por lo que tuvimos que elegir la opción *YES*.

El mensaje dormido del componente Reconfigurar del Agente Sistema al componente *Recuperar_Disp* del Agente Sistema mostrado en la figura 6.33 también fue hecho para corresponder, así que de nueva cuenta, la opción que se tomó fue *YES*.

Los últimos dos casos, mostrados en la figura 6.34 y la figura 6.35 implican el mensaje *activado* para el componente *Reconfigurar_Disp* y para el componente *Recuperar_Disp* ambos del Agente Sistema. Como en los casos anteriores, estos eventos también fueron hechos para corresponder, así que la opción elegida fue *YES* en cada caso.

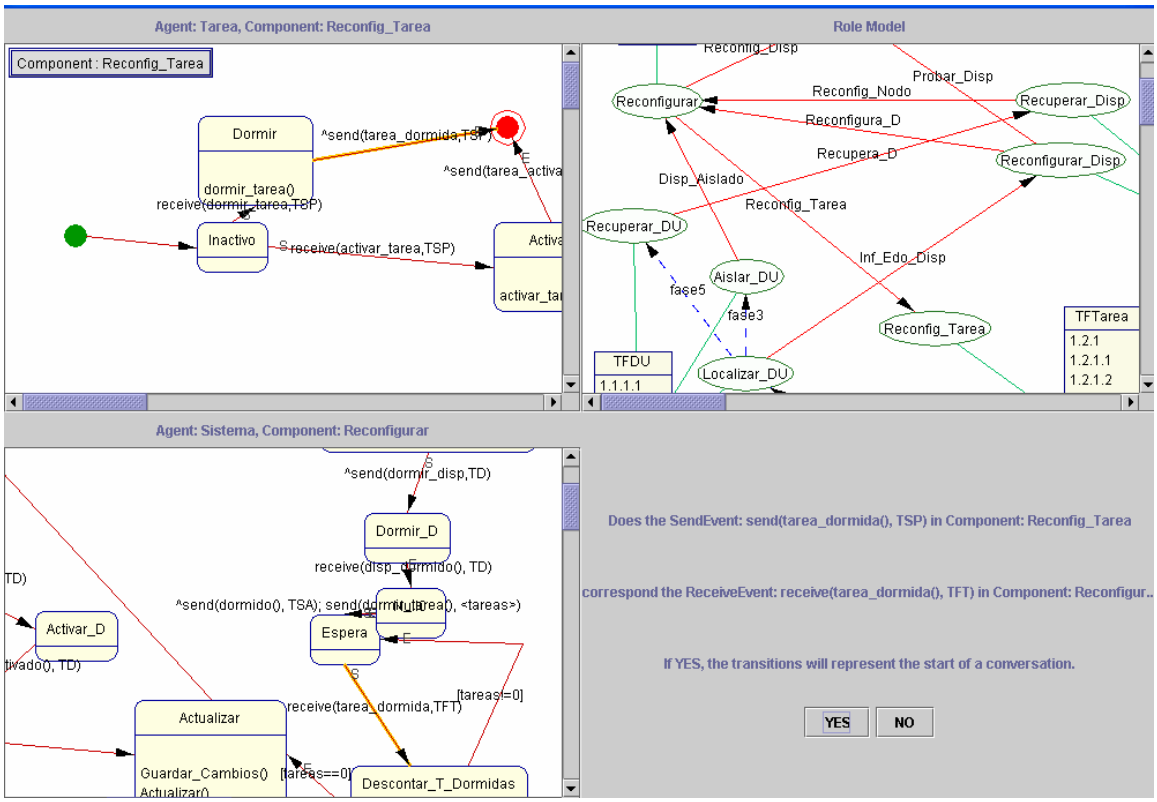


Figura 6. 31 Primer evento de decisión.

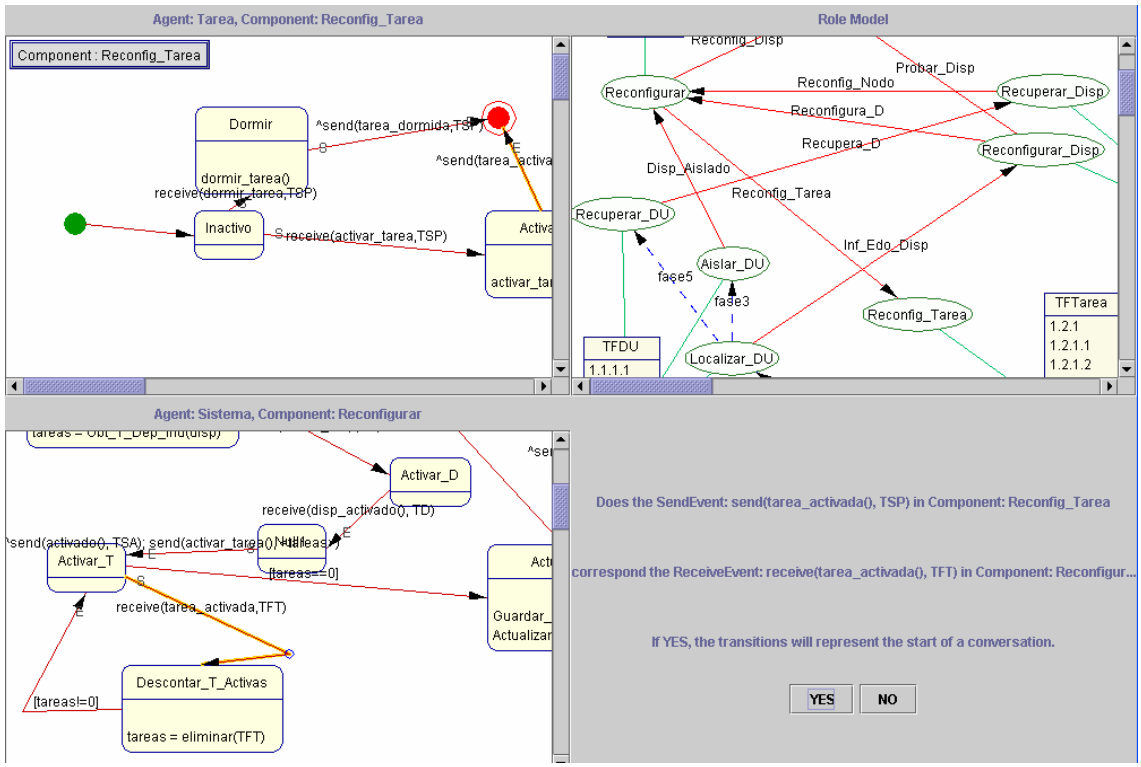


Figura 6. 32 Segundo evento de decisión.

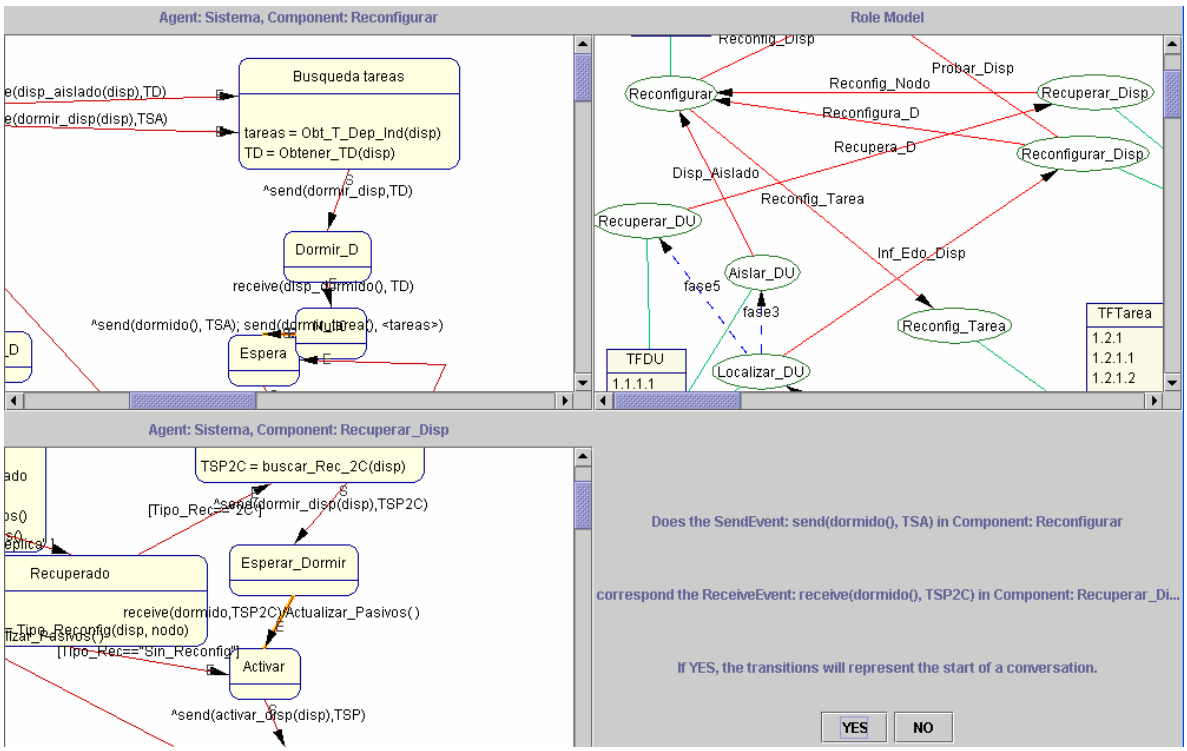


Figura 6. 33 Tercer evento de decisión.

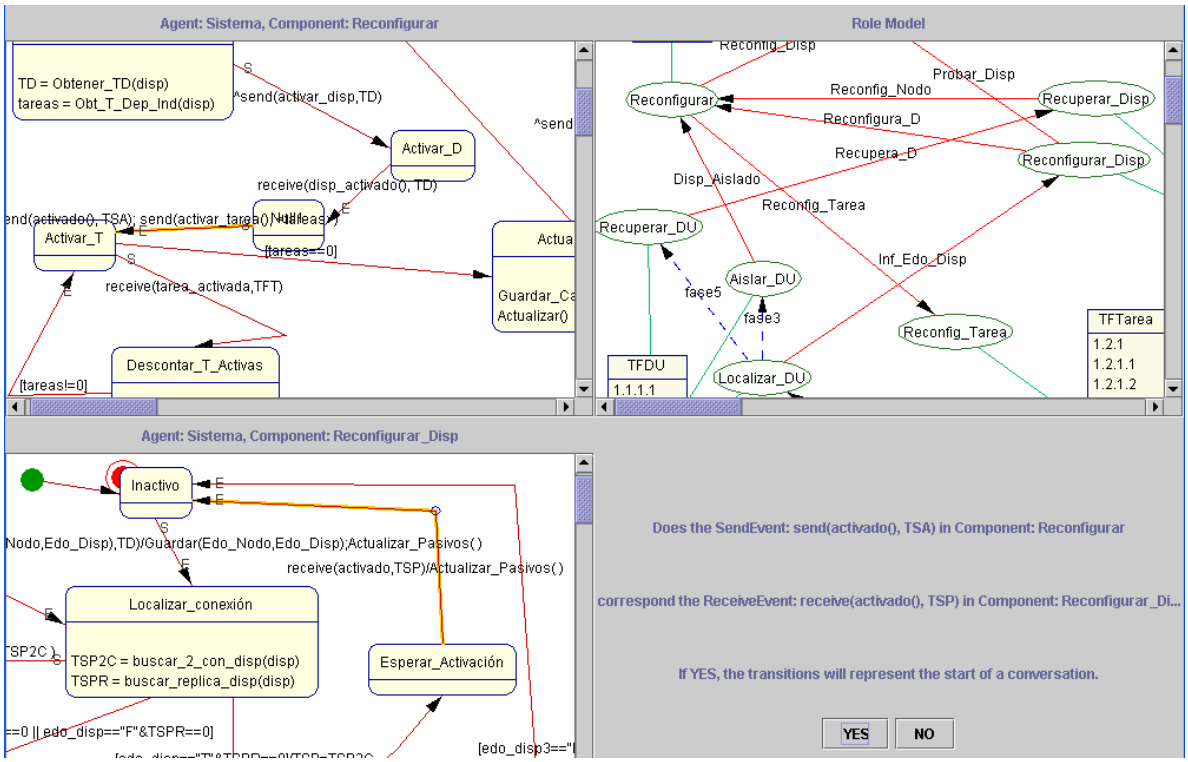


Figura 6. 34 Cuarto evento de decisión.

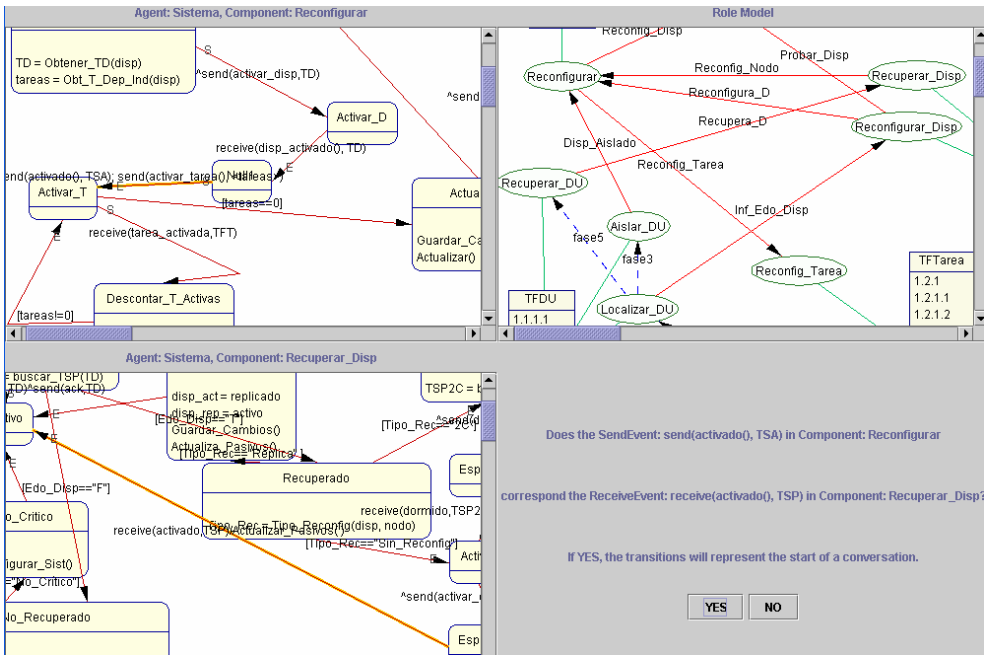


Figura 6.35 Quinto evento de decisión.

6.4.3.2 Anotando los diagramas de estado de los componentes

El resultado de la segunda etapa del proceso de transformación es que todos los diagramas de estado del componente se han anotado, y se ha realizado la concordancia de los eventos que representan el inicio de las conversaciones. El diagrama de estado anotado para el componente *Reconfig_Disp* se muestra en la figura 6.36. La letra "S" al principio de una transición representa donde una conversación iniciará, y la letra "E" en el final de una transición representa el final de una conversación. Los estados y las transiciones entre el inicio y el fin de las etiquetas serán removidos del componente y puestos en los diagramas de estado de la conversación en la etapa siguiente del proceso de transformación. En este caso el componente pertenece directamente a tres conversaciones.

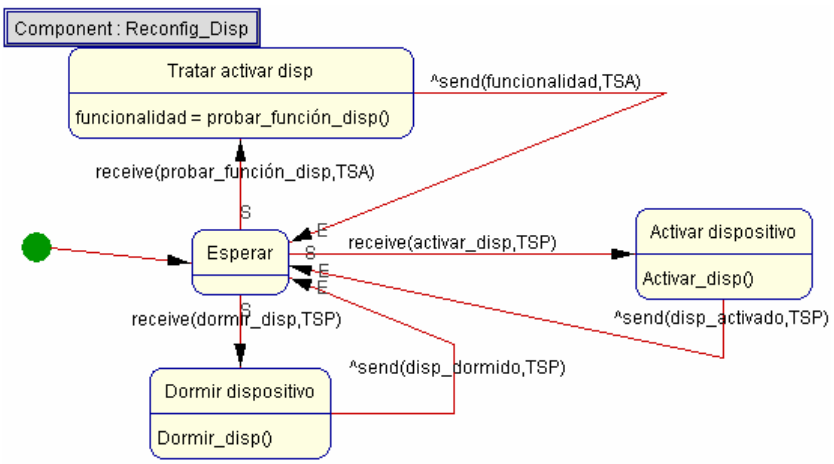


Figura 6. 36 Anotando el componente Reconfig_Disp.

El diagrama de estado anotado para el componente *Localizar_DU* se muestra en la figura 6.37. Una vez más las letras "S" y "E" denotan el principio y el final de las conversaciones que serán removidas de los componentes durante la fase siguiente. Existe el estado nulo que se agregó durante esta etapa del proceso de transformación. El nuevo estado nulo en el diagrama es el resultado de dividir las transiciones que tenían eventos internos (Aislar y Recuperar) y externos (*send(Actualizar(Edo_Nodo,Edo_Disp),TSA)*), que permitieron una clara delineación de donde las conversaciones comienzan y terminan.

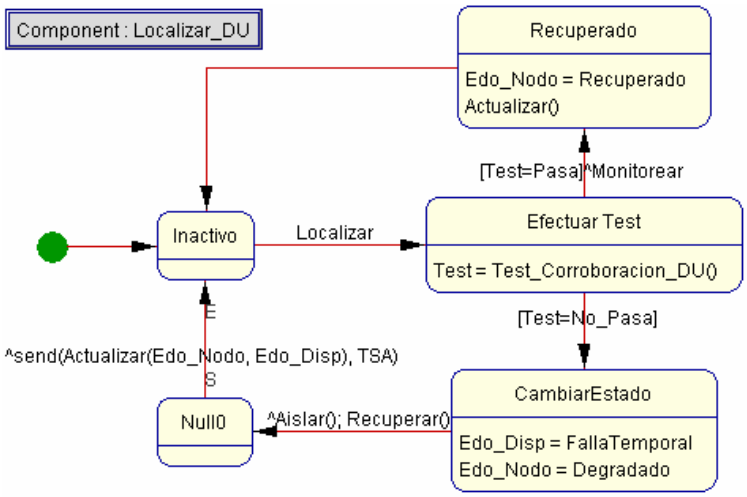


Figura 1 Anotando el componente Localizar_DU.

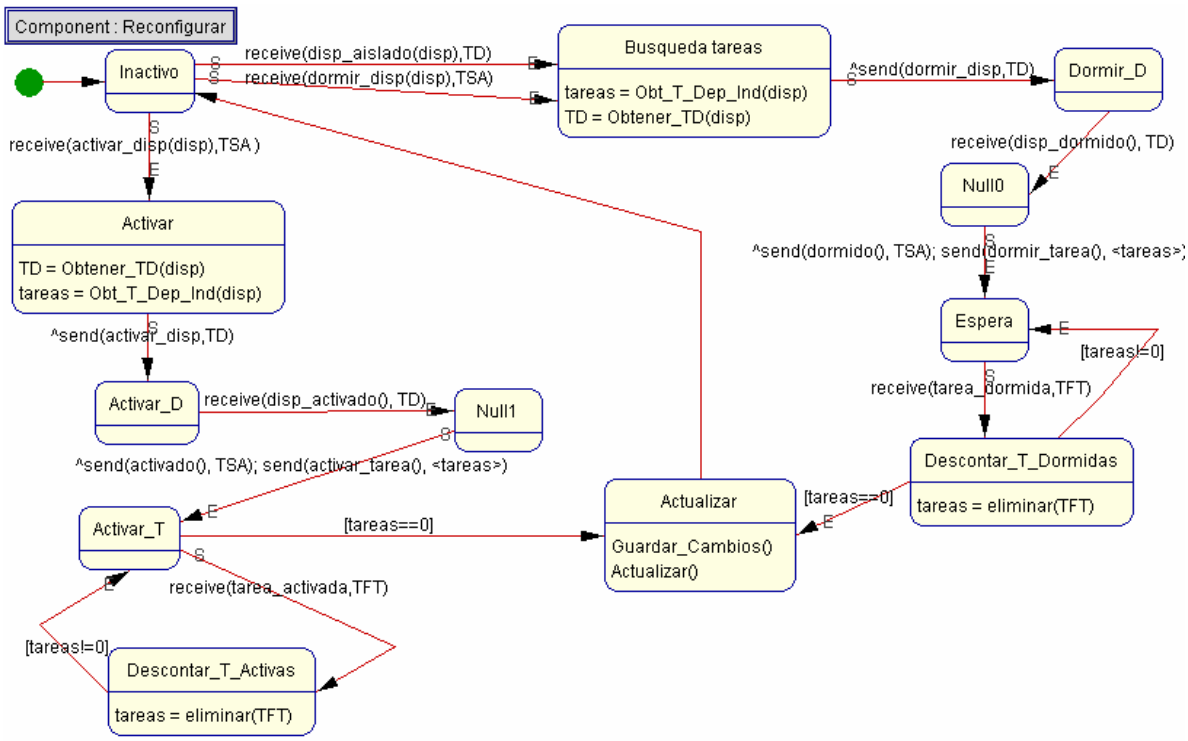


Figura 6. 38 Anotando el componente Reconfigurar.

El diagrama de estado anotado para el componente Reconfigurar se muestra en la figura 6.38. Hay dos nuevos estados nulos agregados durante esta etapa del proceso de transformación ya que dos transiciones contenía tres eventos externos, por lo que fueron separados el evento receive *receive(disp_dormido,TD)*

de los dos eventos send *send(dormido,TSA);send(dormir_tarea,<tareas>)* mediante el estado *Null0*, ya que éste *evento receive* corresponde con la conversación del *evento send* anterior, lo mismo sucede con el estado null1. Hay muchas conversaciones diferentes que emergen de este componente, principalmente debido a los *mensajes multicast* que se envían en el protocolo *Reconfig_Tarea*.

6.4.4 Tercera etapa del proceso de transformación: Create Conversation

En esta etapa, se han anotado todos los diagramas de estado del componente y los mensajes iniciales de las conversaciones anotadas que han sido coincididas. Durante la última etapa del proceso de transformación los diagramas de estados del componente están preparados para removerse de los estados y de las transiciones que pertenecen a las conversaciones. Después se tuvieron que remover y se agregaron a los diagramas de estado las mitades correspondientes de la conversación. Mientras que al remover los componentes se substituyen por una sola transición que tenga una acción que comience la conversación.

La figura 6.39 muestra el diagrama de clase de agente después de que las conversaciones se han agregado entre los agentes. Las transformaciones dieron nombres únicos a las conversaciones.

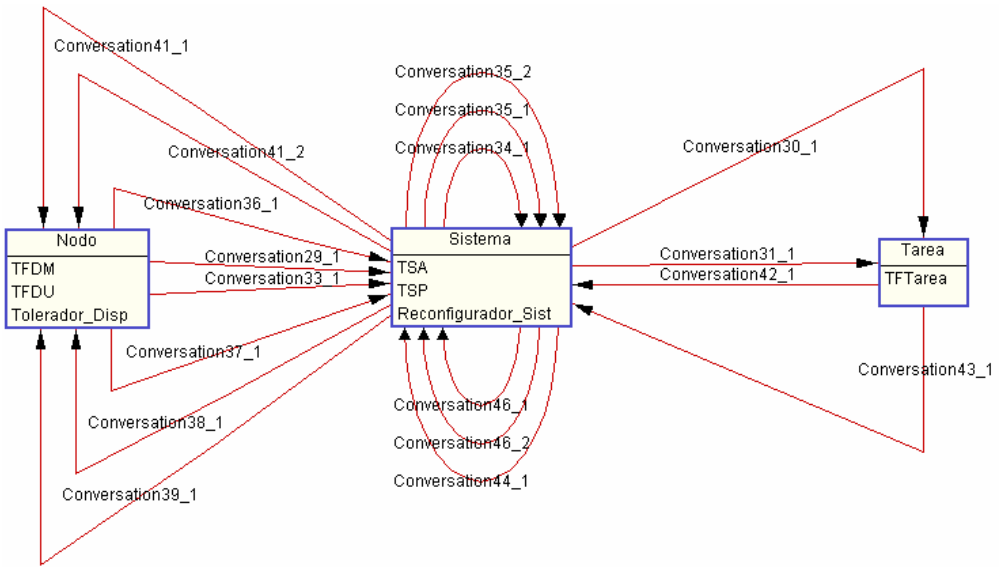


Figura 6. 39 Diagrama de clases de agente con las conversaciones resultantes del proceso de transformación.

El diagrama de estado para el componente *Reconfig_Disp* del agente Nodo después de quitar las conversaciones se muestra en la figura 6.40, el componente *Localizar_DU* del agente, éste mismo agente se muestra en la figura 6.41, por último el componente Reconfigurar del agente Sistema es mostrado en la figura 6.42. Cada diagrama de estado de los componentes define cómo son coordinadas las diferentes conversaciones.

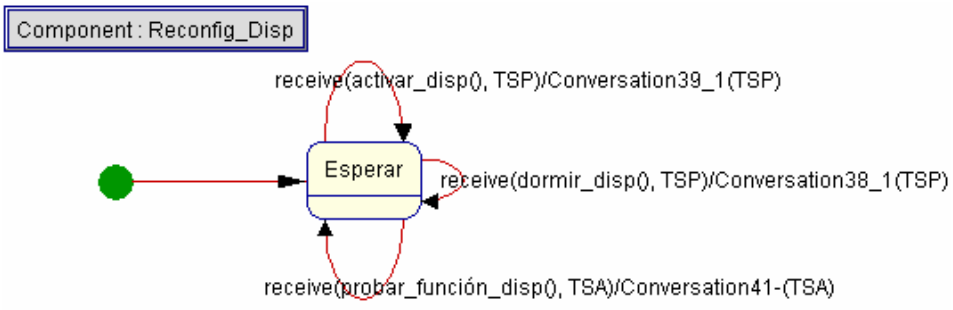


Figura 6. 40 Componente Reconfig_Disp después de la tercera etapa.

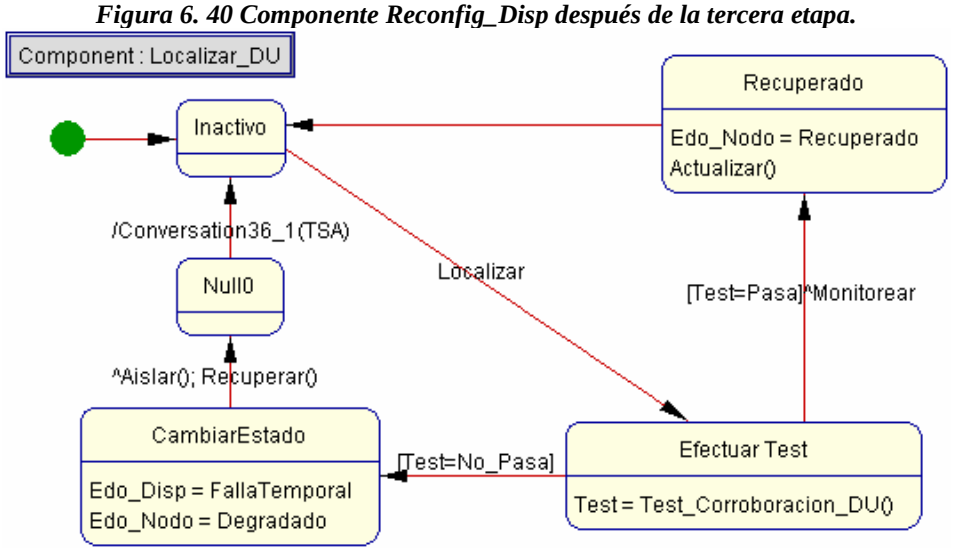


Figura 6. 41 Componente Localizar_DU después de la tercera etapa.

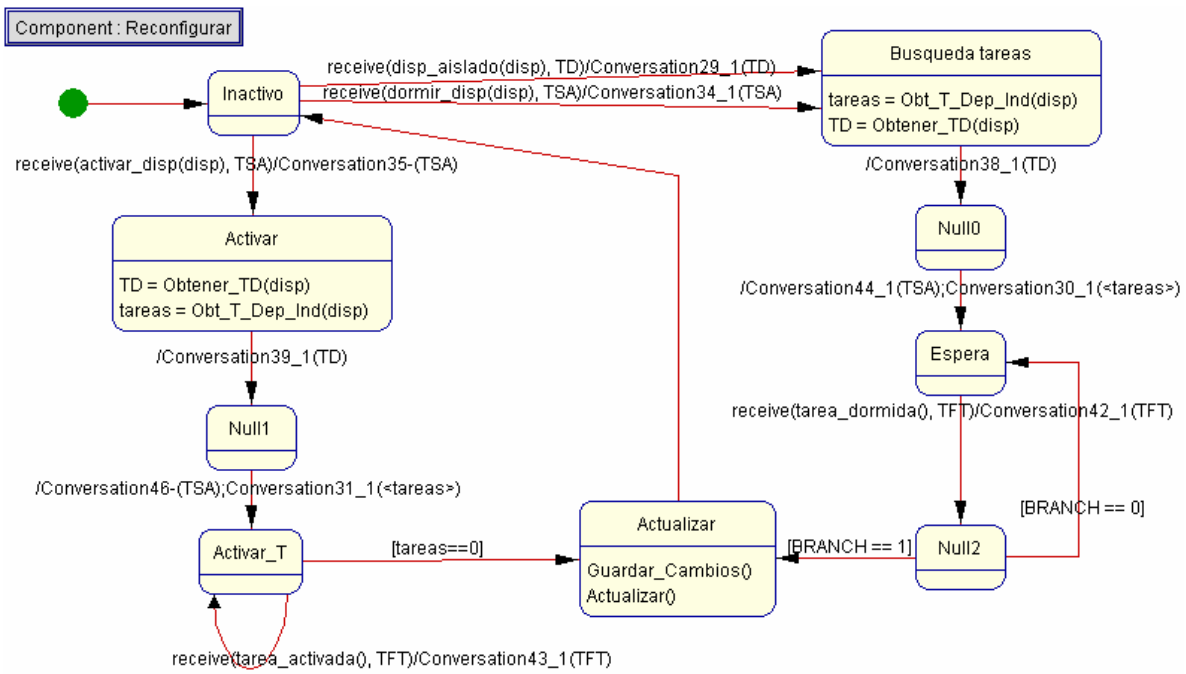


Figura 6. 42 Componente Reconfigurar después de la tercera etapa.

Cada conversación mostrada en el diagrama de clases de agente ahora tiene los estados y las transiciones apropiados del iniciador y el respondedor, pero solamente una conversación será examinada en éste capitulo. La figura 6.43 muestra el diagrama de clase de comunicación para la mitad del iniciador de Conversation41-1. El estado y las transiciones fueron agregados del componente *Reconfigurar_Disp* para crear este diagrama de estado, donde el agente Sistema envía el mensaje *probar_función_disp()* y después recibe el mensaje *funcionalidad()* de parte del agente *Nodo*. Los parámetros *edo_disp* y funcionalidad con *parent* para la acción *edo_disp = funcionalidad* indica que pertenecen al componente del padre (*Reconfig_Disp*), más que a la conversación.

La figura 6.44 muestra el diagrama de clase de comunicación para el respondedor de la conversación Conversation41-1, formada del componente *Reconfig_Disp*. Se recibe el mensaje *probar_función_disp()* y entonces un mensaje de *funcionalidad()* se envía de regreso.

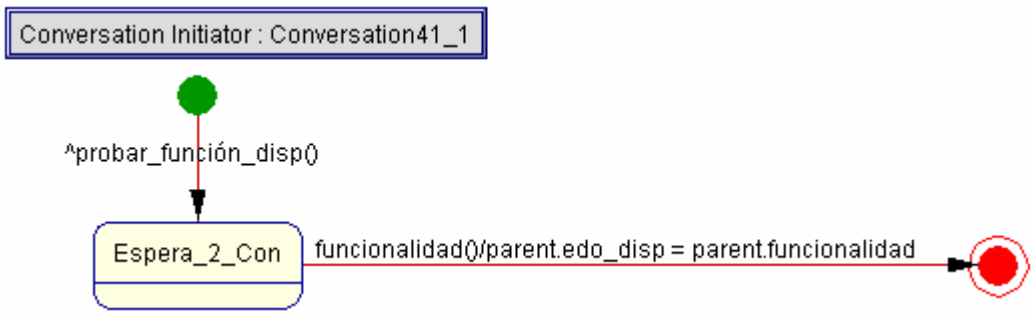


Figura 6. 43 Iniciador de la mitad de la conversación Conversation41-1.

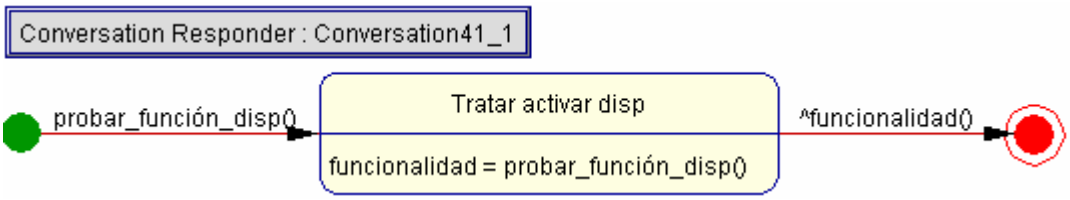


Figura 6. 44 Respondedor de la mitad de la conversación Conversation 41-1.

Al finalizar las tres etapas del proceso de transformación los componentes internos de la arquitectura de los agentes se muestran en la figura 6.45, 6.46 y 6.47 para los Agentes Nodo, Sistema y Tarea respectivamente. La parte superior de cada componente interno representa el nombre del componente, la segunda división contiene los atributos contenidos en el diagrama de estado del componente y la tercera división representa las funciones que se encuentran en el componente. El proceso de transformación crea un proceso para cada conversación relacionada con cada componente, esto es, porque cada conversación al igual que las tareas es ejecutada en un hilo de control. Los atributos y métodos pueden ser eliminados, modificados, incluso se pueden agregar otros.

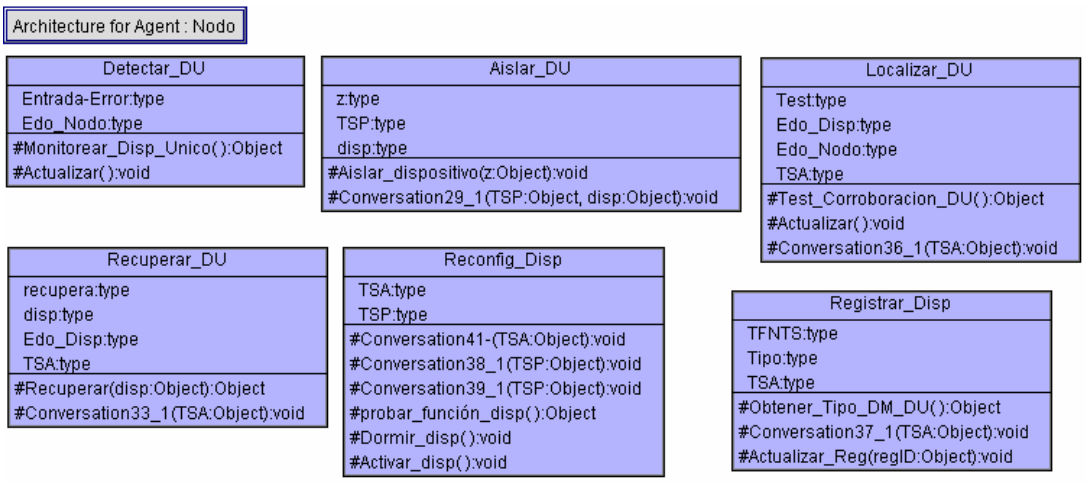
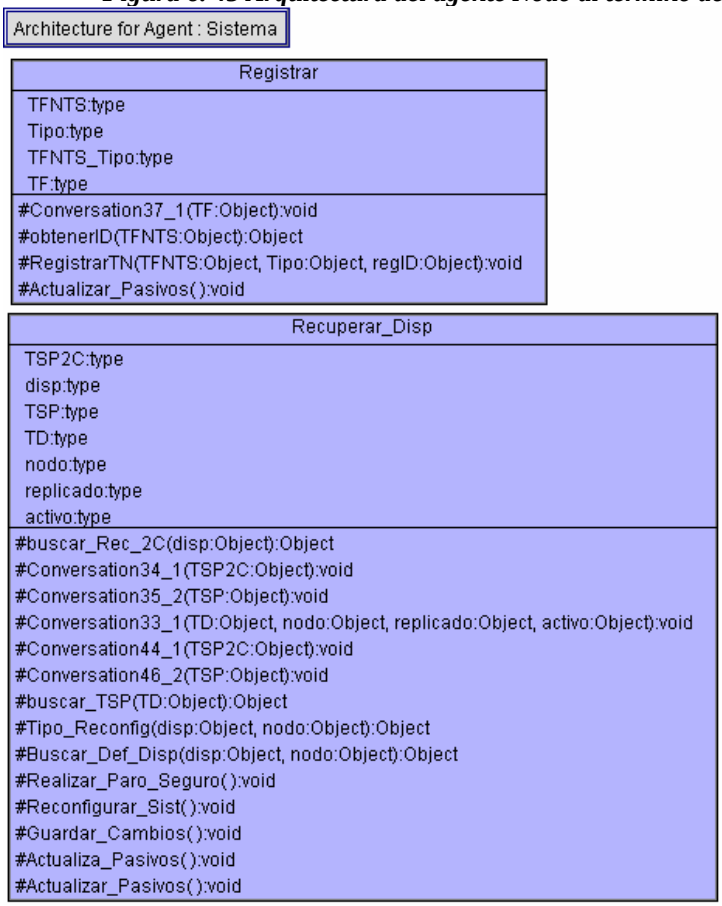


Figura 6. 45 Arquitectura del agente Nodo al término del proceso de transformación.



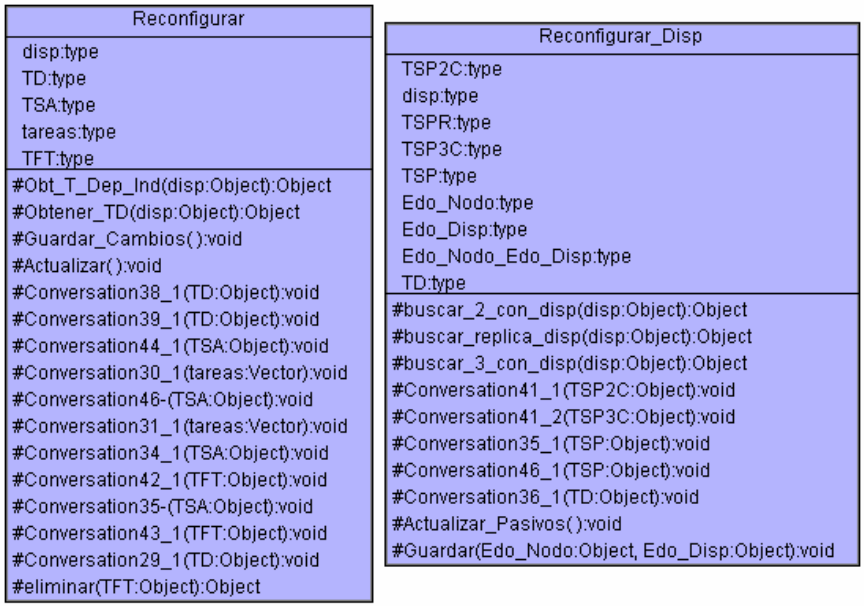


Figura 6. 46 Arquitectura del Agente Sistema al terminar el proceso de transformación.

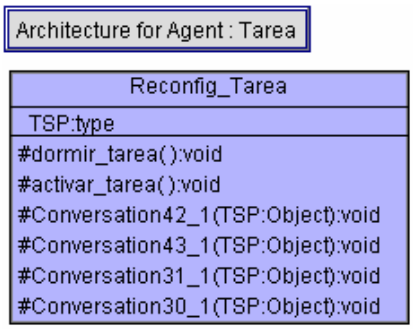


Figura 6. 47 Arquitectura del agente Tarea al término del proceso de transformación.

6.5 Verificación formal de las conversaciones en MaSE

Antes de que un Sistema Multiagente pueda ser confiable según lo esperado, los procesos de comunicación entre los agentes deben ser verificados formalmente.

La herramienta AgentTools provee un módulo que asegura la validez e interoperabilidad de cualquier conversación, AgentTool también en este modulo logra que se pueda cumplir la política del protocolo de comunicación en un determinado sistema propuesto.

Por lo que se introduce un módulo respaldado formalmente mediante algoritmos matemáticos que verifica automáticamente que los protocolos de comunicación definidos en el ambiente de AgentTools haciendo válidos y se comunique según lo esperado antes de que el sistema sea empleado. Esto se puede lograr examinando las conversaciones del agente antes de desplegar el sistema [Lacey 2000].

AgentTool permite crear las conversaciones del agente gráficamente usando diagramas de transición de estados, dado a este conjunto de diagramas la comunicación entre los agentes que puede ser simulada. Usando este acercamiento, las conversaciones se juzgan válidas si la secuencia deseada del mensaje ocurre entre los agentes que se comunican. Este proceso de

juzgar las conversaciones válidas o inválidas se llama *verificación de las conversaciones del agente*. Esta representación gráfica se transforma en un lenguaje de modelado formal llamado Promela (Process Meta Language) que es analizado por la herramienta de verificación Spin para detectar errores tales como callejón sin salida (deadlock), ciclos que no progresan (non-progress loops), errores de sintaxis, mensajes y estados no usados. Una retroalimentación se nos proporciona automáticamente por medio de mensajes de texto y resaltando en las gráficas las condiciones de error que ha encontrado. Promela y Spin vienen integrados automáticamente en el proceso de verificación de conversaciones implementado en AgentTool. Para más información sobre la verificación consultar [Lacey 2000].

Con lo anterior se termina de realizar el modelo que incluye el análisis y diseño del sistema propuesto en este trabajo, bajo la metodología MaSE que nos sirvió de guía en el desarrollo de nuestro sistema. Para verificar que el modelo funciona implementamos un simulador de éste modelo realizado en el lenguaje JADE, dicho simulador y su explicación a detalle y se anexa dentro del CD.

6.6 Conclusiones del capítulo

La construcción del MAS es difícil. Ya que tienen todos los problemas tradicionales de sistemas distribuidos y concurrentes, más las dificultades adicionales que se presentan de requisitos de flexibilidad y de interacciones sofisticadas. Sycara [Sycara 1998] indicaba en 1998 que existían dificultades para trabajar con Sistemas Multiagente. Ya que en ese tiempo no se con una metodología probada que permitiera a los diseñadores estructurar claramente al sistema. Y que tampoco existían herramientas para su desarrollo que fueran flexibles con las numerosas características que tienen los agentes.

Sin embargo actualmente se puede contar con la varias metodologías para el diseño de MAS. Aunque aún no existe una que sea común. Se cuenta con varias que fueron mencionadas en el Capítulo 2 y que nos pueden ayudar a modelar SMA de alta calidad como es el caso de la metodología MaSE seleccionada para desarrollar el SMA tolerante a fallos en este trabajo.

Modelar nuestro Sistema Multiagente con la metodología MaSE fue relativamente no muy complicado aunque si muy minucioso, y se requirió de mucha concentración. Esta metodología fue de gran ayuda desde el análisis hasta la implementación del simulador. La utilización de metas desde el principio del modelado nos centra desde el inicio de lo que se quiere realizar, y nos marca la pauta para obtener un SMA bien definido, fiable, flexible, y validado completamente.

Cuando desarrollamos el simulador fue relativamente fácil, y funcionó de acuerdo al modelo ya que éste cumplió en su totalidad con los requerimientos que nos propusimos al inicio de este trabajo, lo pudimos lograr debido a que definimos de una manera correcta nuestros requerimientos, y seguimos los pasos que la metodología MaSE nos indicó utilizando como soporte su herramienta de desarrollo. Por lo que creemos que es de vital importancia la utilización de está u otra metodología de desarrollo para obtener que nuestro Sistema Multiagente funcione de acuerdo a lo establecido por nuestros requerimientos. Los casos de metas y los diagramas de secuencia fueron vitales para describir perfectamente nuestro sistema. Los roles en el caso partícula me sirvieron mucho para modelar éste sistema, fueron de vital ayuda para definir las tareas y poder visualizar como ir desarrollando el sistema sin perderme, el diseño de las tareas y las comunicaciones utilizando autómatas finitos ayudan definitivamente de una manera formal a explicar la función especifica que tiene cada agente. Todo esto lo pudimos lograr ya que se definió puntualmente los requerimientos del sistema

Capítulo 7 Conclusiones y trabajos futuros

Una vez presentada la arquitectura tolerante a fallos a nivel hardware y software para el sistema de control de un robot móvil, modelada como un Sistema Multiagente, y realizado el simulador que la valida, su utilidad para otros trabajos relacionados y su evaluación se presentan en las conclusiones y aportaciones de éste trabajo así como las líneas de investigaciones futuras.

7.1 Conclusiones

La solución de los problemas de detección, aislamiento y recuperación de fallos en los robots móviles requiere la integración de varias metodologías. Los investigadores en este tema comentan que el tratamiento de fallos en los sistemas robóticos móviles esta ligado a la arquitectura utilizada en la navegación, y los mecanismos tolerantes a fallos como son: detección, aislamiento y recuperación no están tan aisladas como sucede en los sistemas industriales.

La investigación inicia introduciéndonos al mundo de los robots, a la tecnología de agentes y Sistemas Multiagente, y sus diferentes metodologías como herramienta para analizar y diseñar el sistema que es la base de la arquitectura tolerante a fallos a nivel hardware y software para el sistema de control de un robot móvil que se presenta en éste trabajo. En la actualidad, existen diferentes métodos para tolerar los fallos en los sistemas de control en robots móviles de una manera parcial, pero no existen arquitecturas bien definidas y fiable que ofrezcan tolerar fallos de una manera integral, en este trabajo se modela y simula a través de un Sistema Multiagente una arquitectura que tolera los fallos de una manera integral y relativamente simple en el sistema de control de un robot móvil.

Las tres contribuciones principales de esta tesis son:

Se realiza una nueva manera de tolerar los fallos proporcionando mecanismos basados en la filosofía dividir y vencer.

El rediseño de la arquitectura física de control distribuido, donde se realizan dobles conexiones de actuadores y sensores para que la tolerancia a fallos del el microcontrolador, controlador de la red y memorias inmersos en cada nodo que conforma esta arquitectura, se realice de una manera flexible.

Se realizan una modificación a la arquitectura software 3+, de tal manera que el sistema de control del robot pueda soportar esta nueva arquitectura tolerante a fallos.

7.2 Resultados destacables

A continuación se resumen los resultados más destacables, agrupados en diferentes secciones.

7.2.1 Arquitectura distribuida

La arquitectura que aquí se propone se compone de una red de agentes distribuidos que cooperar, se coordinan y se comunican para ejecutar mecanismos de tolerancia a fallos en cada componente (ya sea de hardware o software) que integran el sistema de control del robot. Al realizar esto se obtiene una solución flexible y adaptable de tolerar fallos en el sistema de control del robot de una manera integral.

7.2.2 Análisis de fiabilidad

Se realiza un análisis de fiabilidad de mediante árboles de fallos difusos con el objetivo de establecer los modos de fallos de los sensores, actuadores y así determinar

7.2.3Agentes para cada componente

Cada agente que constituye la arquitectura tolerante a fallos se encarga de supervisar, a cada sensor, actuador, tarea, microcontrolador, etc. que integra el sistema de control, de tal manera trata únicamente situaciones de fallo durante el funcionamiento del robot, presentando las siguientes innovaciones:

Formalización de una arquitectura jerárquica de agentes para la monitorización robusta de todos los componentes que integran el sistema de control.

Conjunto original de supervisores (agentes) para realizar los mecanismos tolerantes a fallos de una manera relativamente sencilla durante el funcionamiento del robot móvil.

La arquitectura es abierta, de tal manera que los mecanismos tolerantes a fallos de cada componente pueden ser adaptables, ya que son piezas de software intercambiables.

La conexión redundante de actuadores y sensores en diferentes nodos, ayuda a reconfigurar el sistema de control ante un fallo del microcontrolador, memorias o controlador de la red que pudiera suceder en algún nodo que integra el sistema distribuido de control de robot, los agentes pueden realizar dicha reconfiguración del fallo de una manera sencilla, ya que su funcionamiento lo pueden seguir realizando en el nodo donde están doblemente conectados, de tal manera que solo se degrada el sistema por un solo componente que falló, y no por todos los demás elementos que integran al nodo, y que no están en modo de fallo.

La bitácora de fallos que se genera y se almacena en un archivo, es de una gran ayuda cuando se realiza un paro del robot ya sea por reparación o por mantenimiento programado, ya que el técnico cuenta con información respecto al momento que ocurrió un fallo, en que dispositivo, si se recuperó o no, facilitando así su trabajo.

El sistema orientado agentes, que se representa, entre otras cosas es de gran funcionalidad tolerando los fallos de una manera integral.

Los requisitos necesarios presentados en el Capítulo 4, se modela mediante la metodología MaSE en el Capítulo 6, y se valida mediante el simulador realizado en JADE, que nos da como resultado la afirmación que el sistema cumple con los requisitos establecidos.

7.3 Trabajo y líneas de investigación futuras

Éste trabajo de investigación sienta las bases de un sistema integral y distribuido para tolerar fallos en un sistema de control para un robot móvil, dejando abiertas diferentes líneas de investigación, para completar, mejorar lo ya existente y para desarrollar nuevos elementos. A continuación se reseñan algunas de las líneas inmediatas relacionadas con los temas discutidos en éste trabajo.

1.- La arquitectura aquí tratada no causa ningún problema, pero podría hacerlo al ser aplicada a un caso general.
2.- La arquitectura trabaja en sistemas totalmente distribuidos, podría rediseñarse para realizarlo en sistemas parcialmente distribuidos.
3.- Esta arquitectura se realizó para el control de un robot móvil, pudiéndose probarse en otros sistemas de control.
4.- La implementación de la inteligencia de los agentes es sencilla, podría realizarse utilizando una red neuronal para que estos puedan aprender y razonar y así lograr un mayor grado de autonomía.
5.- Se pueden realizar algoritmos difusos para la detección de fallos donde los datos para realizar la detección de los sensores y actuadores manejen posibilidades.
6.- Realizar análisis de fiabilidad utilizando cadenas de Markov difusas para obtener un mayor éxito al diseñar los algoritmos de detección y aislamiento de los fallos en los diversos componentes que integran al sistema robótico.

Referencias

Arnulfo Alanis Garza, Juan José Serrano, Ors Carot. (2004). Intelligence Agents in Distrubuted Faul Tolerant System for Industrial Control, Proceedingd of the Intenational Conference on Artificial Intelligence, IC-AI'04, Vol 1, Las Vegas, Nevada, USA, June 21-24, CSREA Press, ISBN: 1-932415-31-9.

Arnulfo Alanis Garza, Juan Jose Serrano Ors Carot. (2005) Intelligence Agents in Distrubuted Faul Tolerant System, Proceedingd of the Intenational Conference on FNG 2005, International Conference on Fuzzy System, Neural Network and Genetic Algorithms, FNG'2005, Instituto de Tijuana, Mexico, Octubre 13-14, CIS Chapter IEEE

Arai S., Sycara K. Payne T. R. (2000). "Multi-agent Reinforcement Learning for Scheduling Multiple-Goals," in *Proceedings of the Fourth International Conference on Multi-Agent Systems* (ICMAS'2000).

Athanázio de Cerqueira Gatti Máira, Pereida de Lucena Carlos José (2007). An Agent-Based Approach for Building Biological Systems: Improving the Software Engineering for Complex and Adaptative Multi-Agent Systems ISSN 0103-9741 Monografias en Ciencias Computacionales Número 14/07

Attouche, S., Hayat S. y Staroswiecki M. (2001). An e_cient algorithm for the design of fault tolerant multi-sensor system. Proceedings of the 40th IEEE Conference on Decision and Control 2, 1891-1892.

Bazzan Ana (2005).A Distributed Approach for Coordination of Traffic Signal Agents. In *Autonomous Agents and Multiagent Systems*, 10(2): 131-164.

Berlanga A., Borrajo D., Fernández F., García R., Molina J. M., Sanchis A. (1999). "Robótica y agentes inteligentes para su control y manipulación".

Bernhard Kaiser, Peter Liggesmeyer, Oliver Mäckel (2005). A New Component Concept for Fault Trees.

Blanke, M. (2000). Fault-tolerant control systems. In: New Trends in Advanced Control. Springer-Verlag.

Blanke, M., Kinnaert M. Lunze J., Staroswiecki M. (2003). Diagnosis and fault-tolerant control. Springer-Verlag. Germany.

Bonarini, A., G. Bontempi (1994). A qualitative simulation approach to fuzzy dynamical models. ACM Transactions on Modeling and Computer Simulation (TOMACS) 4, 258-313.

Bonasso A., Peter R., Kortenkamp D. y Miller D. (1995). 'Experiences with An Architecture forIntelligent, Reactive Agents', INTELLIGENT AGENTS II: Agent.

Boris Gnedenko, Igor Ushakov. (1995). Probabilistic Reliability Engineering. John Wiley & Sons.

Bratman M. (1997). What is intention? In Intentions in Communication, MIT Press, Cambridge, pp 15- 31, Cohen P.R., Morgan J., and Pollack M.E. (Eds). .

Brazie, F.M.T, (1997). Dunin-Keplicz, Jennings, N.R, Treur, J., DESIRE: Modelling multiagente systems in compitional formal framework.

Brooks, R.A.(1986). A Roboust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation RA-2, 14-23.

Camacho D., Borrajo, J. M. Cammarata S., McArthur D., Steeb R. (2004). Strategies of Cooperation in Distributed Problem Solving, en Readings in Distributed Artificial Intelligence, Ed. Alan H. Bond and Les Gasser, Morgan Kaufmann.

Cassandras, C.G., Lafortune S., Olsder G.J. (1995). Introduction to the modelling, control and optimisation of discrete event systems. In: Trends in Control (A. Isidori, Ed.). Springer-Verlag.

Conry S., Meyer R.A., Lesser V.(1988). Multistage Negotiation in Distributed Artificial Intelligence.

Carbó J., Molina J. M, Dávila J. (2003). Trust Management through Fuzzy Reputation. International Journal of Cooperative Information Systems. vol 12, nº 1 pp 135-155, Marzo 2003.

Carvallo, Walter, Leuschen (2004). Interval Methods for Fault-Tree Analyses in Robotics
Carlos Carreras, *Member IEEE* Universidad Politécnica de Madrid, Madrid, Ian D. Walker, *Member IEEE* Clemson University, Clemson.

Corchado J.M. (2003).Constructing Deliberative Agents with Case-based Reasoning Technology. International Journal of Inteligent Systems. Vol 18, No. 12, December. ISSN 0884- 8173.

Corchado Juan Manuel. (2003). Una puerta hacia la convergencia de la inteligencia artificial. Departamento de Informática y Automática Facultad de Ciencias Universidad de Salamanca, apuntes.

Corchado J. M., Pavón P., Corchado E. S., Castillo L. F. (2004). Development of CBR-BDI Agents: A Tourist Guide Application. European Conference on Case-based Reasoning 2004. Springer Verlag.

Cox E. (1992). Fuzzy Fundamentals IEEE Spectrum, October, pp. 58-61.

Chellas B. F. (1980) Modal Logic: an introduction. Cambridge University Press, Cambridge.

Chen, J., Patton R.J., Chen Z. (1998). An approach to fault-tolerant control of uncertain systems. Held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Proceedings of the 1998 IEEE International Symposium on Intelligent Systems and Semiotics (ISAS) pp. 175-180.

Chen, J., Patton R.J. (1999). Robust Model- Based Fault Diagnosis for Dynamic Systems. Kluwer Academic Publishers.

Chow, E.Y., Willsky, A.S. (1984). Analytical redundancy and the design of robust failure detection systems. IEEE Transactions on Automatic Control AC-29, 603-614.

Clark, R.N. (1989). State estimation for instrument fault detection. In: Fault diagnosis in dynamic systems, theory and application (Clark RN Patton RJ, Frank PM, Ed.). Prentice Hall. New York.

Dardinier-Maron, V., Hamelin F. y Noura H. (1999). A fault-tolerant control design against major actuator failures: application to a three-tank system. Proceedings of the 38th IEEE Conference on Decision and Control 4, 3569-3574.

De Kleer, J. y Brown, J.S. (1984). Qualitaitve physics based on confuences. Artificial Intelligence 24, 77-83.

DeLoach, S (1999). Multiagent Systems Engineering a Methodology and Lenguage for Designing Agent systems, agents Oriented Information Systems (AOIS'99), May 1, Seattle Washington.

DeLoach, S. (2001) *Analysis and Design using MaSE and agentTool*. Actas de conferencia. Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS).

Ding, X., Jeinsch, T. (1999). An approach to analysis and design of observer and parity relation based fdi systems. Proceedings of XIV IFAC World Congress.

Emami-Naeini, A., Akhter M.M., Rock S.M. (1988). Efect of model uncertainty on failure detection: the threshold selector. IEEE Transactions on Automatic Control AC-33, 1106-1115.

EURESCOM (2000). MESSAGE: Methodology for engineering systems of software agents. Initial methodology. Technical Report P907-D1, EURESCOM.

Félix Ingrand (2003). Architectures Logicielles pour la Robotique Autonome, Journées Nationales de Rechercher en Robotique, Octubre, 2003.

Fernández L. Joaquín. (1998). Supervision, detection, diagnosis and exception recovery in autonomous mobile robots, Departamento de Ingeniería de Sistemas y Automática. Escuela Superior de Ingenieros Industriales De la Universidad De vigo España.

Fernández L. Joaquín. (2002). A hierarchical approach to detect different kinds of failures in a robotic system. In Proceedings of Conference on Intelligent Robotics and Systems, Vancouver, Canada, October 2002.

Ferrell Cinthya (1999). Failure Recognition and Fault Tolerance of an Autonomous Robot. Thesis of Master MIT 1999.

FIPA. (2003). Foundation for Intelligent Physical Agents.

Frank, P.M., Ding S.X., Köppen-Seliger B. (2000). Current developments in the theory of fdi. IFAC Symposium SAFEPROCESS 1, 16-27.

G. Caire, F. Garijo, J. Gomez, J. Pavon, E. Vargas (2000). Agent Oriented Analysis using MESSAGE/UML.

G. Caire, M. Cossentino, A. Negri, A. Poggi, and P. Turci (2004).Multi-agent systems implementation and testing. In *From Agent Theory to Agent Implementation –Fourt International Symposium (AT2AI-4)*, Vienna, Austria, April. 3.3

Glaser N.(1996). Contribution to Knowledge Modelling in a Multi_Agent Framework (the CoMoMAS approach). PhD Thesis L' Universitité Henri Poincaré, Nancy I, France, November.

Georgeff. M.(1998). Rational Software Agents: From Theory to Practice. In Agent Technology: Foundations, applications and markets. Jennings & Wooldridge (Eds). Pag. 139-160. Springer.

Gertler, J.J.(1991). Analytical redundancy methods in fault detection and isolation. IFAC Symposium SAFEPROCESS 1, 9-21.

Giampapa J.A., Sycara K.(2003). Conversational Case-Based Planning for Agent Team Coordination, Robotics Institute, Carnegie Mellon University.

Giret B., Insfrán, Pastor, Cernuzzi.(2000).Informe técnico OO.Method Para desarrollo de sistemas multiagentes, Departamento de sistemas infórmaticos y computación, Universidad Politécnica de Valencia, 29 junio

Giret B. Adriana (2005). Anemonia: *Una Metodología Multi-Agente para sistemas Holónicos de Fabricación*, Tesis Doctoral, de sistemas infórmaticos y computación de Universidad Politécnica de Valencia, España

Glez-Bedia M, Corchado J. M., Corchado E. S. and Fyfe C.(2002). Analytical Model for Constructing Deliberative Agents, Engineering Intelligent Systems, Vol 3: 173-185.

Glez-Bedia M., Corchado J. M. (2003). Constructing autonomous distributed systems using CBR-BDI agents. Innovation in knowledge Engineering. Faucher C., Jain L. and Ichalkaranje N. (eds.).

Gomez Sanz Jorge. (2004). *Modelado de Sistemas Multi-gente*. Tesis, Facultad de Informática, Universidad Complutense.

Grosz B. y Sidner C.(1990) Plans for discourse. Intentions for Communication, pp 417-444, MIT Press.

Haddadi A.,Sundermeyer K. (1996) Belief-Desire-Intention Agent Architectures´, Foundations of Distributed Artificial Intelligence, pp 169-185, O´Hare G. M. P.; Jennings N. R. (Eds) Wiley-Interscience Publication.

Herrin, S.A. (1981). Maintainability applications using the matrix fmea technique. IEEE Transactions R-30 pp. 212-217.

Horak, D.T., Guidance, J. (1988). Failure detection in dynamic systems with modelling errors. Control and Dynamics 11(6), 508-516.

Hoblos, G., Staroswiecki M., Aitouche A. (2000). Optimal design of fault tolerant sensor networks. Proceedings of the 2000 IEEE International Conference on Control Applications pp. 467-472.

Huo, Y., Ioannou P., Mirmirani M. (2001). Faulttolerant control and reconfguration for high performance aircraft: Review. CATT Technical Report 01-11-01. University of Southern California Dept. of Electrical Engineering- Systems Los Angeles, CA 90089 California State University, Los Angeles Dept. of Mechanical Engineering.

Iglesias F. C. Angel (2000).Thesis Doctoral Una metodología para análisis y diseño de sistemas multi-agantes basada en conocimiento, MAS- Coommon KADS.

Jacobson I., Christerson , M., Jonson P., Overgaard G. (1999). “Object-Oriented Software Engineering”. A Use case Driven Approach ACM Press, 1999.

Jensen K, (1999) Coloured Petri Nets: High Level Language For Systems Design and Analysis. Berlin Heidelberg Springer 1999.

Jennings N. R. (1996). Coordination tecniques for distributed artificial intelligence. In: O’HARE, G.M.P.; JENNINGS, N.R. (Eds.). Foundations of distributed artificial intelligence. New York: John Wiley & Sons, 1996. p.187-210.

Jennings N. R., Wooldridge M. (2000). Agent-oriented software engineering. *Handbook of Agent Technology* (ed. J. Bradshaw). AAAI/MIT Press.

Jennings, N. R. (2001). Building complex, distributed systems: The case for an agent-based approach. *Comms. of the ACM (Special issue on agents in telecomms),*
44(4):35–41.

Joanne Bechta Dugan, Stacy A. Doyle. (1996).New Results in Fault-Tree Analysis. In Proceedings Annual Reliability and Maintainability Symposium Tutorial Notes, pages 1-23.

John D. Healy (1996).Basic Reliability. In Proceedings Annual Reliability and Maintainability Symposium Tutorial Notes, pages 1-16.

Julian Vicente, Botti Vicente (2004). Estudio de métodos de desarrollo de sistemas multiagente, Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. No.18 (2003), pp. 65-80. ISSN: 1137-3601. © AEPIA (http://www.aepia.org/revista).

Jung W.S, Yang J.E., Ha J. (2006). Development of measures to estimate truncation error in fault tree analysis [An article from: Reliability Engineering and System Safety].

Kaehler Steven D. (2004). Fuzzy Logic Introduction Part 2 2004

Kazerooni H. (2006). Design, Control Systems, Robotics, Human-Machine Systems, Manufacturing Machines. Massachusetts Institute of Technology.

Keith D., Pannu A., Sycara K. y Williamson M. (1997) ‘Designing Behaviors for Information Agents’, AUTONOMOUS AGENTS ’97, Proceedings of the First International Conference on Autonomous Agents, pp 404-413, Marina del Rey CA, Feb 5-8, 1997, ACM Press.

Kendall, Zhao (1998).Role Modeling for Component Design And Modeling Transport Objects with Patterns. JOOP 10(8): 26-32 (1998).

Kinny D., Georgeff. M. (1997) Commitment and effectiveness of situated agents. Proceedings of Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93) Chambery, France, 1993.

Kinny D. (2000) A methodology and modelling technique for Systems of BDI agents. Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW96). Lecture Notes in Artificial Intelligence 1038. 1996.

Kuipers B. (2006) An intellectual history of the Spatial Semantic Hierarchy. In Margaret Jefferies and Wai K. Yeap (Eds.), Robot and Cognitive Approaches to Spatial Mapping,

Lacey Timothy H. Captain, (2000). USAF, A Formal Methodology and Technique for Verifying Communication Protocols in Multiagents Environment thesis AFIT/GCS/ENG/00M-12, Air Force Institute Technology Base Ohio.

Lakos Charles (1995). From coloured Petri nets to object Petri nets. In proceedings of the "Aplication and Theory of Petri Nets 1995", volumen 935 of LCNC, pages 278-297, Torino Italy, June 1995 Springer.

Lee C.C. (1990).Fuzzy Logic in Control Systems EEE Trans. on Systems, Man, and Cybernetics, SMC, Vol. 20, No. 2, 1990, pp. 404-35.

Leitch R., Shen, Q. Conghil G.-Chantler M. y Slater A. (1994). Qualitative model-based diagnosis of dynamic systems. Colloquim of the Institution of Measurement and Control.

Leuschen Martin (1997). Robot Reliability Through Fuzzy Markov Models, Thesis for the Degree Master of Science, University Rice

Lind, J. (1999). MASSIVE: Software Engineering for Multi-agent Systems. PhD thesis, DFKI. Alemania.

Lyshevski, S., Dunipace K.,Colgren R. (1999).Identifcation and reconfigurable control of multivariable aircraft. Proceedings of the American Control Conference.

Maes P. (1989) 'Situated Agents Can Have Goals', Designing Autonomous Agents: Theory and Practice From Biology to Engineering and Back, pp 49-71, Maes, Pattie, (Ed), 1989.

Maes P. (1995) "Artificial Life Meets Etertainment: Life like Autonomous Agents ", Communications of the ACM, vol 38 No 11, pp 108-114, 1 995

Malone T. W. (1988) "What is coordination theory?" National Science Foundation Coordination Theory Workshop, MIT.

Mariusz Nowostawski, Martin Purvis, "A Layered Approach for Modelling Agent Conversations"Stephen Cranefield Department of Information Science University of Otago, Dunedin, New Zealand.

Marks Christopher, Captain, USAF, Extensible Multi-Agent System for Heterogeneous Database Association Rule Mining and Unification Thesis Presented to the faculty of the Graduate School of Engineering of the Air Force Institute of Technology Air University. 1999.

Martínez B, Humberto. 2001 *Una Arquitectura Distribuida para el Control de Robots Autónomos Móviles: un Enfoque Aplicado a la Construcción de la Plataforma Quaky-Ant,* Tesis Doctoral, Departamento de Ingeniería de la Información y las Comunicaciones, Universidad de Murcia, España.

Maciejowski (2001). Invariant Sets for Constrained Nonlinear Discrete-time.

Meléndez J., Colomer J. (2003). Introducing Qualitative Reasoning in Fault Dictionary Techniques. Monográfico. Data Mining**.** Ed. Red Española de Minería de Datos y Aprendizaje Automático.

Michitaka Oshima (2006). A prototype of fault diagnostic system for robots. International conference on Industrial and engineering applications of artificial intelligence and expert system. Proceedings of the 2nd international conference on Industrial and engineering applications of artificial intelligence and expert systems Volume 1

Minsky, M. 1975 A framework for Representation Knowledge Mc Hill. New York.

Molina. (2004) “Intelligent Travel Planning: A Multiagent Planning System to Solve Web Problems in the e-Tourism Domain”, Autonomous Agents and Multi- Agent Systems. 4 (4), pp 387-392, Diciembre 2001.

Moraes Márcia Ctristina, da Rocha Costa Carlos (2003). Estruturando o Espaço de Coordenação de Sistemas Multiagentes IV Encontro Nacional de Inteligencia Artificial Campinas ,SP, Brasil 2003.

Morari, M., Baotic M. y Borrelli F. (2003). Hybrid systems modeling and control. European Journal of Control 9, 177-189.

Mosquera Fontán Celestino, “Análisi y studio experimental de herramientas basadas en agents”, Proyecto de fin de carrera, facultad de infórmatica Universidad Da Coruña, julio 2001.

Müller J. P. (1997) The Design of Intelligence Agents (LNAI 1037). Springer-Verlag: Berlin, Germany, 1997.

Murilo Juchem, Melo Bastos Ricardo (2002). Projetando Sistemas Multiagentes em Organizações Empresariais Gramado, RS, Brasil – 2002 XVI Simpósio Brasileiro de Engenharia de Software.

Newell, A. & Simon, H. A. (19821), Computer Science as Empirical Inquiry: Symbols and Search, in J. Haugeland, ed., 'Mind Design', MIT Press, Cambridge, Massachusetts, chapter 1, pp. 35{66.

Nodine Marian H., Unruh Amy Constructing Robust Conversation Policies in Dynamic Agent Communities, Microelectronics and Computer Technology Corporation (MCC) Austin, Texas.

Muñoz Martinez Víctor F. (2004). Control de movimiento en un robor quirurguico a prueba de fallos.XXV Jornadas de Automática, Ciudad Real Septiembre 8 2004.

Nwana H.S. (1996) Software Agent Technologies. British Telecommunications Tecnilogy Journal 14,1996.

Odell, J., Parunak, H., and Bauer, B.(2000). Extending UML for agents. In Proceedings of the Agent-Oriented Information Systems Workshop, pages 3–17.

Odell, J., Parunak, H., and Bauer, B. (2001). Representing agent interaction protocols in UML. In Proceedings of the AGENTS’2000, Barcelona, Spain.

Oliveira, D. and Ferreira, P. R. and Bazzan, A. L. C. (2004). Aplicação de abordagens baseadas na organização de colônias de insetos sociais para resoluçãoo de problemas de organização em Sistemas Multiagentes 344.

OMG (2001).Unified Modeling Language Specification (Draft). Version 1.4, February 2001

Pacheco Sánchez José A. (2005). Arquitectura Software para un Sistema Robótico Móvil. Tesis de maestría Instituto Tecnologico de Monterrey Campus Morelos.

Patton, R. J. (1997). Fault-tolerant control: the1997 situation. Proceedings of IFAC Symposium on SAFEPROCESS pp. 1033_1055.

Posadas Yagë Juan L. (2003). Arquitectura para el control de robots móviles mediante delegación de código y agentes, Tesis doctoral, Departamento de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia España.

Pressman, R. S. (1982). *Software Engineering: A Practitioner's Approach*. Libro completo. McGraw-Hill Series in Software Engineering and Technology. McGraw-Hill, Inc.1982.

Puig, V., Quevedo J., Escobet T.,De las Heras S. (2002a). Robust fault detection approaches using interval models. IFAC World Congress.

Puig, V., Quevedo J. (2002b). Passive robust fault-detection using fuzzy parity equations. Mathematics and computers in Simulation 60, 193-207.

Puig, V., Saludes J., Quevedo J. (2003). Worst-case simulation of discrete linear timeinvariant interval dynamic systems. Reliable Computing 9(4), 251_290.

Puig, V., Quevedo, J., Escobet, T., Morcego, B.; Ocampo, C. (2004). Control Tolerante a Fallos (Parte I): Fundamentos y Diagnóstico de Fallos. *Revista iberoamericana de automática e informática industrial*, 1 (1) :15-31. ISSN: 1697-7912

Rao A. S., Georgeff M. P. (1991) Modeling Rational Agents within a BDI-Architecture. Second International Conference on Principles of Knowledge Representation and Reasoning (KR91). San Mateo, C.A. 1991.

Rao A. S. and Georgeff M. P. (1992) An abstract architecture for rational agents. In Proceedings of Third International Conference on Principles of Knowledge Representation and Reasoning Morgan Kaufmann Publishers, San Mateo, C.A. 1992.

Rao A. S. y Georgeff M. P. (1995) BDI Agents from Theory to Practice. Proceedings of the First International Conference on Multi-Agents Systems. (ICMAS-95). San Francisco. June 1995.

Rao A. S. y Georgeff M. P. (1998) Decision procedures of BDI. Logics. Journal of logic and computation 8(3). 1998.

Raphael M.J. y DeLoach S. A knowledge base for Knoeledge- Based Multiagent System Construction *National Aerospace and Electronics Conference (NAECON)*
Dayton, OH, October 10-12, 2000.

Raphael M.J. knowledge base Support for Desing and Synthesis of Multiagents Systems, Thesis Department of the Air Force Air University, Air Force Institute of Technology, March 2000.

Reynolds C. (1987) “Flocks, Herds and Schools: A Distributed Behavioral Model”. Computer Graphics. 21 (4) pp. 25-34.

Ricordel, P. M. (2001) *Programmation Orientée Multi-Agents , Développement et Déploiement de Systèmes Multi-Agents Voyelles*. Tesis doctoral. INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE.

Rising Linda, Pattern Almanac 2000, Ed. Addison-Wesley, 2000.

Robinson David J. 1st Lieutenant, USAF. A Componen Based Approach to Agent Specification**,** Thesis AFIT/GCS/ENG/00M-22, Air Force institute Technology, Base Ohaio, 2002

Rosenschein S. and Kaelbling, L. P. (1986) The synthesis of digital machines with provable epistemic properties. *In J.Y.Halpen, editor. Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning about Knowledge. Morgan Kaufmann, 1986.*

Rumbaugh, J. Blaha, M. Premerlani, W., Eddy, F,. and Lorensen, V,. "Object-Oriented Modeling and D esingN". Prentice Hall, 1995.

Salido, M., Barber, F. (2002) A Polynomial Algorithm for Continuous Non-binary Disjunctive CSPs.

Sandholm T., Lesser V. R. (1995) "Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Protocol". *Proc. of the Second International Conference on Multiagent Systems, 328-335. Menol Park, California, AAAI Press.*

Shen, Q. y Leitch, R. (1993). Fuzzy qualitative simulation. IEEE Transactions on SMC.

Schild K. (1999) On the relation between standard BDI logics and standard logics of concurrence. *Proceedings of the Fifth International Workshop on Agent Thories, Architectures and Languages (ATAL-98), Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1999.*

Shaley D.M., Tiran J. (2007). Condition-based fault tree analysis (CBFTA): A new method for improved fault tree analysis (FTA), reliability and safety calculations [An article from: Reliability Engineering and System Safety])

Seel N. (1989) Agents Theories and architectures. PhD. Thesis. Surrey University, Guildford, UK. 1989.

Self Athie Captain, USAF Desing and Specification of Dynamic, Mobile, and Reconfigurable Multiagent Systems Thesis AFIT/GCS/ENG/01M-11, Air Force Institute of Technonogy, Ohaio.

Singh M.P., (1994) "Multiagent Systems: A Theoretical Framework for Intentions, Know How and Communications", Lecture Notes in Artificial Intelligence 799, Springer Verlag.

Smith R.G., (1980) "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", IEEE Trans. On Computers, vol 29, nº12, pp 1104-1113.

Sparkman C.H., DeLoach (2001). S. Automated Derivation of Complex agent architectures fom Analysis Specifications, *Second International Workshop On Agent-Oriented Software Engineering (AOSE-2001) Montreal, Canada - May 29th 2001*

Sparkman C.H 1st Lieutenant, USAF (2002)Tranforming Analysis Models Into Desing Models for the Multiagent Systems Engineering MaSe Metthodology Thesis AFIT/GCS/ENG/01M-12, Air Force Institute technology Base Ohio.

Staroswiecki, M., Hoblos G. y Aitouche A. (2004). Sensor network design for fault tolerant estimation. International Journal of Adaptive Control and Signal Processing p. in press.

Steel L., (1989) "Cooperation between Distributed Agents through Self-Organization", Decentralized AI, Demazeau, Muller (ed.) Elsevier.

Steel L., (1994) "A Case Study in the Behavior-Oriented Design of Autonomous Agents", *Proc. Of the '94 Conference on Simulation of Adaptive Behavior*, MIT Pres.

Suthankar G., Sycara K. (2004) Team-aware Robotic Demining Agents for Military Simulation, Robotics Institute, Carnegie Mellon University.

Sutton Ian (2004). Faul Trre Analysis eBook ESBN: C60-802C-2c45-72D1

Shaw Mary y Garlan David, Software Architecture: Perspectives on an emerging discipline, Ed. Prentice Hall, 1996.

Sycara K. (1989) "Multiagent Compromise via Negotiation". En *Distributed Artificial Intelligence*, Editores Gasser y Huns, Pitman, 1989.

Sycara K. (1997) "Using Option Pricing to Value Commitment Flexibility in Multiagent Systems.Technical Report CMU-CS-TR-97-169, School of Computer Science, Carnegie Mellon University.

Sycara K., Decker, K., Pannu A., Williamson M., and Zeng D. (1996) "Distributed Intelligent Agents." IEEE Expert: Intelligent Systems and Their Applications. Vol. 11, No. 6, Dec., 1996, pp. 36-46.

Sycara K., Paoluci M., van Velson M. and Giampapa J. The RETSINA MAS Infraestructure, Robotics Institute, Carnegie Mellon University.

Tao, G., Chen Sh. y Joshi S.M. (2002). An adaptive control scheme for systems with unknown actuator failures. Automatica 38, 1027-1034.

Von Martial F. (2001).The Intelligent Software Agents Group, The Robotics Institute, Carnegie Mellon University, RETSINA Coordinating Plants of Autonomous Agents. Lecture Notes in Artificial Intelligence No. 610. Springer-Verlag, Berlin

Vazquez (2003). Apuntes del ingeniero Alejandro J.G. Vásquez Nava para el ITESM en ñla materia impartida de agentes y sistemas multiagentes en el Campus Monterrey.

Vesely, W. E., Goldberg, F. F., Roberts, N. H.,. Haasl, D. F. (1981). *Fault Tree Handbook.* U. S. Nuclear Regulatory Commission, NUREG-0492, Washington

Visinsky Monica L. (2003). Fault Detection and Fault Tolerance Methods for Robotics Tesis Doctoral, Universidad de Rice.

Walker Ian D., Visinsky M. L., Cavallaro J. R. (1991). Fault Detection and Fault Tolerance in Robotics. In Proceedings of NASA Space Operations, Applications, Research Symp., Houston, TX, July 1991. In Press.

Walker I. D., Cavallaro J. R(1996). Fault Tolerant Robotic Architectures and Algorithms. In SIAM International Conference on Industrial and Applied Math- ematics, Washington, DC, July.

Walke Ian., Carrera Carlos (2005) Interval Methods for Fault-Tree Analyses in Robotics. Publication Date: Mar 2005 Volume: 50, Issue: 1 On page(s): 3-11 ISSN: 0018-9529

Wang, J. y Shao, H. (2000). Delay-dependent robust and reliable h1 control for uncertain time-delay systems with actuator failures. Journal of the Franklin Institute 337, 781-791.

Wavish P. R. and Connah D. M. (1990) "Representing Multi-Agent Worlds in ABLE". Technical Note, TN2964, Philips Research Laboratories.

Wen Chun Yaun Cai Kai Yuan, Zhang Ming Lian (1992). Fuzzy variables as a basis for a theory of fuzzy Reliability in the Possibility ontest. Fuzzy Sets and Systems, 42(2):145{172.

Wesson R. et all "Network Structures for Distributed Situation Assessment", Readings in Distributed Artificial Intelligence, Ed. Alan H. Bond and Les Gasser, Morgan Kaufmann 1988.

Wood, M., DeLoach, S. (2000). *Developing Multiagent Systems with agentTool*. Actas de conferencia. ATAL 2000. LNAI 1986. Castelfranchi, C. and Lespérance, Y.2000.

Wood, Mark F. Captain, USAF (2000). Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems Thesis AFIT/GCS/ENG/00M-26, Air Force Institute Technology Base Ohio.

Wood, M. y DeLoach, S.(2001). In *Agent-Oriented Software Engineering – Proceedings of the First International Workshop on Agent-Oriented Software Engineering*, 10th June 2000, Limerick, Ireland. P. Ciancarini, M. Wooldridge, (Eds.) Lecture Notes in Computer Science. Vol. 1957, Springer Verlag, Berlin, January 2001.

Wooldridge M. (1994) 'Agent Theories, Architectures and Languages: A Survey', Intelligent Agents (ECAI-94 Workshop Proceedings on Agent Theories,

Wooldridge, M., Jennings, N. R. Kinny, D., (1996). *The Gaia Methodology for system multiagents.*

Wooldridge M. (1997). "Agent-based software engineering," *Proc. Inst. Elec. Eng.*, vol. 144, pp. 26-37.

Wu W., Rao S.S. 2006.) Uncertainty analysis and allocation of joint tolerances in robot manipulators based on interval analysis [An article from: Reliability Engineering and System Safety] [HTML] (Digital)

Zhang, Frank, P.M. (2000). Qualitative observer and its application to fault detection and isolation systems. Systems and Control Engineering 211, 253-262.

Zhang, Y., Jiang, J. (2003). Bibliographycal review on recon_gurable fault-tolerant control systems. Proceedings IFAC SAFEPROCESS pp. 265-276.

Zhou, Yu-guo, Zhang Ying-wei y Wang Fu-li (2002). A new type of recon_gurable control against actuator faults. Proceedings of the 4th World Congress on Intelligent Control and Automation 4, 2710-2713.

Zlotkin G., Rosenschein J. S. (1993) One, two, many coalitions in multi-agent systems. Proceedings of the Fifth European Workshop on Modelling Autonomous Agentes in a Multi-Agent world (Ghedira and K. Sprumont, eds.), U. of Neuchatel, Switzerland.

Apéndice A

Caso de meta: Tolerador de Fallos a Nivel Nodo en un Dispositivo Múltiple

a) Objetivo: Tolerar los fallos de los dispositivos múltiples que conforman el nodo, ya sea actuadores o sensores.
b) Precondiciones: El tolerador a fallos nivel nodo, nivel tarea y nivel sistema deben estar listos al iniciar el sistema. El sistema de control del robot este listo y correcto para trabajar.
c) Iniciador: El Tolerador a Fallos Nivel Nodo en Dispositivo Múltiple inicia a monitorear al dispositivo, y el sistema de control debe estar corriendo.
d) Post-condiciones: El Tolerador a Fallos Nivel Nodo en Dispositivo Múltiple detectó fallo en alguno de los dispositivos múltiples o están libre de fallos
e) Roles:

- Tolerador de Fallos Nivel Nodo en Dispositivo Múltiple (TDM)
- Tolerador de Fallos a Nivel Tarea (TFT)
- Tolerador de Fallos a Nivel Sistema Activo (TSA)
- Tolerador de Fallos a Nivel Sistema Pasivo (TSP)

f) Descripción del Rol:

1. Inicia cuando se activa el sistema de control del robot y todas las entradas de error están libre de error.
2. El TDM le informa al TSA que le dé de alta. El TSA lo registra, se actualiza y actualiza a todos los TSP.
3. El TDM inicia su monitoreo en el dispositivo, éste puede detectar o no un fallo en el dispositivo:
   a) Si el dispositivo esta libre de error continúa monitoreándolo.
   b) Si detecta fallo en el dispositivo pone al nodo en estado sospechoso, se lo comunica al TSA; el TSA se actualiza y le informa los cambios a todos los TSP, entra a la fase de localización.
4. En la fase de localización efectúa un test de corroboración de fallo a los diferentes dispositivos que integran al dispositivo múltiple y localiza el dispositivo que ha fallado; al realizar éste test a cada dispositivo, éste puede obtener dos hechos:
   a) El dispositivo pasó la prueba: Al pasar el dispositivo la prueba de corroboración, el TDM continúa aplicando el test a los demás dispositivos. Si todos los dispositivos pasan el test continua monitoreando, pone al nodo en estado recuperado, se lo comunica al TSA, el TSA se actualiza, envía un mensaje de actualización a todos los TSP.
   b) El dispositivo no pasó la prueba: Al no pasar un dispositivo la prueba de corroboración el TDM lo pone en falla temporal (solo a ese dispositivo), se lo comunica al TSA, este se actualiza y se lo comunica a todos los TSP; pasa a la fase de aislamiento (paso 5) y a la fase de recuperación (paso 7).
5. En la fase de aislamiento el TDM realiza las siguientes acciones:

   - El TDM realiza una serie de acciones para aislar el dispositivo fallido.
   - El TDM descuenta a dicho dispositivo del número de dispositivos sin fallo.
   - El TDM compara el número de dispositivos sin fallos que le restan con un mínimo de dispositivos que se establece de ante mano con el que pueda seguir trabajando el dispositivo de manera confiable. Después de comparar los dispositivos pueden suceder los siguientes dos casos:
     a) La cantidad de dispositivos activos es mayor o igual al mínimo permisible (puede seguir trabajando el dispositivo múltiple).

- El TDM reconfigura al dispositivo múltiple en el nodo para trabajar con los dispositivos que aún le quedan activos y continúa monitoreando al dispositivo múltiple.
- El TDM de dicho dispositivo múltiple le comunica al TSA que se ha reconfigurado el dispositivo múltiple, el TSA se actualiza, actualiza a todos los TSP.

b) La cantidad de dispositivos activos es menor que el tope permisible (no puede seguir trabajando el dispositivo múltiple).

- El TDM del nodo aísla la señal del dispositivo múltiple, pone al dispositivo múltiple en fallo temporal y al nodo en estado degradado, se lo comunica al TSA, éste se actualiza, actualiza a todos los TSP y pasa a la fase de reconfiguración.
- El TDM le informa a su TSP que ha aislado al dispositivo múltiple y que el tope es menor al permisible. Cuando el TSP localizado en el nodo en el que se encuentra el TDM se percata del fallo del dispositivo realiza una búsqueda de todas las tareas que dependen directamente o indirectamente del dispositivo que falló, se comunica con los TFT de estas tareas informándoles que deben dormir su tarea respectiva, igualmente le comunica al TDM que debe dormir la señal del dispositivo.
- El TDM y los TFT duermen su dispositivo o tarea según el caso y se lo comunican a su TSP.
- El TSP se actualiza y actualiza al TSA, el TSA actualiza a todos los TSP.

6. En la fase de reconfiguración cuando el TDU le envía al TSA un mensaje de actualización en la fase de aislamiento, el TSA al percatarse del fallo del dispositivo múltiple realiza los siguientes casos:

- El TSA busca en su B.C. el nodo que contiene la doble conexión de este dispositivo (supongamos se encuentra en el nodo 2), para cerciorarse del fallo del dispositivo o de su conexión en caso de funcionar en su doble conexión; de la misma manera busca el nodo que contiene la réplica del dispositivo múltiple (en caso de existir), pongamos como ejemplo que su replica se encuentra en el nodo 3. Pueden acontecer los siguientes casos:

a) El TSA envía un mensaje al TDM del nodo 2 que contiene la doble conexión del dispositivo que falló, en dicho mensaje le informa que pruebe si funciona el dispositivo.

- El TDM del nodo 2 realiza un procedimiento de verificación para cerciorarse del fallo del dispositivo.
- El TDM le indica al TSA el resultado del procedimiento; es decir si el dispositivo funciona o no en ese nodo.
- En caso de funcionar el dispositivo, el TDM espera a que el TSA le informe si deberá activar al dispositivo o no; estas acciones se especifican en el punto 6.c.1.

b) Existe réplica de dicho dispositivo.

- El TSA le comunica al TSP del nodo 3 en el cual se encuentra la réplica del dispositivo que falló, que debe reconfigurar al dispositivo y sus tareas.
- El TSP busca en su B.C. todas las tareas que dependen directamente o indirectamente del dispositivo, se comunica con los TFT de estas tareas informándoles que activen su tarea respectiva y le informa al TDM del dispositivo replicado que active a su dispositivo.
- El TDM y los TFT activan al dispositivo y tarea respectivos.
- El TDM y los TFT le comunican al TSP del nodo 3 que han activado su dispositivo o tarea según sea el caso.

- El TSP se actualiza, actualiza al TSA, éste actualiza a todos los TSP.

c) No existe réplica de dicho dispositivo. En este caso puede acontecer lo siguiente:

1. El dispositivo funciona en el nodo 2, el cual contiene la doble conexión:

    - El TSA al recibir el mensaje por parte del TDM del nodo 2 en el que le indica que el dispositivo funciona y al percatarse que no existe réplica de dicho dispositivo le comunica al TSP del nodo 2 el cual contiene la doble conexión del dispositivo, que debe reconfigurar al dispositivo y sus tareas.
    - El TSP del nodo 2 busca en su B.C. todas las tareas que dependen directamente o indirectamente del dispositivo, se comunica con los TFT de estas tareas informándoles que activen su tarea respectiva y le informa al TDM de la doble conexión del dispositivo que active a su dispositivo.
    - El TDM y los TFT activan su tarea y dispositivo respectivos.
    - El TDM y los TFT le comunican a su TSP (TSP2) que han activado su dispositivo y tarea respectivamente.
    - El TSP2 se actualiza y actualiza al TSA, el TSA actualiza a todos los TSP.

2. El dispositivo no funciona en el nodo 2.

    - El TSA al recibir el mensaje por parte del TDM del nodo 2 en el que le indica que el dispositivo no funciona y al percatarse que no existe réplica de dicho dispositivo busca en su B.C. si existe otra doble conexión en un tercer nodo; si existe se realizan las mismas acciones que en el paso 6.b.a, solo que si funciona el dispositivo lo activa.

3. El dispositivo no funciona en su doble conexión y no existe réplica de dicho dispositivo.

    - Al no funcionar el dispositivo en los nodos que contienen su doble conexión y al no contar con réplica de este dispositivo, el TSA espera un momento determinado a que el TDM en el nodo 1 le indique que ha recuperado al dispositivo, en este caso realiza las acciones definidas en el punto 7.b.2.c.
    - Si el TDM le indica al TSA que no ha recuperado el dispositivo en el nodo 1, el TSA realiza las acciones del punto 7.a.

7. Posteriormente se entra a la fase de recuperación, en la que se trata de recuperar al dispositivo fallido; en esta fase se tienen los siguientes casos:

a) El dispositivo que ha fallado no ha sido recuperado.

1. Si este dispositivo al fallar no sobrepasó el mínimo permisible de dispositivos activos.

    - El TDM del nodo 1 le informa al TSA que dicho dispositivo se debe poner en baja permanente. El TSA se actualiza, actualiza a todos los TSP.
    - El sistema continúa funcionando de acuerdo a las acciones tomadas en la reconfiguración.

2. Si este dispositivo al recuperarse y al darlo de alta en los dispositivos activos por el TDM del nodo 1 es menor al límite permisible:

    - El TDM le informa al TSA que el dispositivo múltiple no ha sido recuperado, el TSA se actualiza, actualiza a todos los TSP.
    - El TSA busca en su B.C. si dicho dispositivo es crítico o no es crítico.

- En caso de ser un dispositivo crítico el TSA realiza un paro seguro del sistema.
- En caso de no ser crítico el dispositivo el TSA continúa con las acciones tomadas en la fase de reconfiguración en caso de haberse activado el dispositivo en algún nodo; en caso contrario el TSA reconfigura al sistema sin dicho dispositivo.

b) Se recupera el dispositivo que falló.

1. Si este dispositivo al fallar no sobrepasó el mínimo permisible de dispositivos activos para que el dispositivo múltiple continúe su operación.

- El TDM del nodo 1 lo suma al número de dispositivos sin fallos.
- El TDM reconfigura al dispositivo recuperado con el dispositivo múltiple.
- El TDM se comunica con el TSA indicándoles que se ha recuperado dicho dispositivo y se ha reconfigurado al dispositivo múltiple.

2. Si este dispositivo al recuperarse y al darlo de alta en los dispositivos activos por el TDM del nodo 1 es igual al límite permisible; para esto se requiere realizar alguna de las siguiente acciones según sea el caso:

a) Cuando el sistema se reconfiguró usando la doble conexión de este conjunto de dispositivos en un nodo 2.

- El TSA al percatarse de la recuperación del dispositivo le comunica al TSP en el nodo 2 (doble conexión) que duerma al dispositivo y las tareas.
- El TSP del nodo 2 busca en su B.C. todas las tareas que dependen directamente o indirectamente del dispositivo, se comunica con los TFT de estas tareas informándoles que deben dormir su tarea respectiva y le informa al TDM de la doble conexión del dispositivo que duerma a su dispositivo.
- El TDM y los TFT duermen su dispositivo y tareas respectivamente.
- EL TDM y los TFT le comunican al TSP del nodo 2 que han dormido su dispositivo y tarea respectivamente.
- El TSP se actualiza y actualiza al TSA, el TSA se actualiza a todos los TSP.
- El TSA al recibir el mensaje por parte del TSP del nodo 2 de que ha dormido al dispositivo y tareas le comunica al TSP1 que active al dispositivo y tareas en el nodo.
- El TSP del nodo 1 busca en su B.C. todas las tareas que dependen directamente o indirectamente del dispositivo, se comunica con los TFT de estas tareas informándoles que activen su tarea respectiva y le informa al TDM de la doble conexión del dispositivo que active a su dispositivo.
- EL TDM y los TFT le comunican al TSP del nodo 1 que han activado su dispositivo y tarea respectivamente.
- El TSP1 se actualiza y actualiza al TSA, el TSA se actualiza a todos los TSP.

b) Cuando el sistema se reconfiguró usando la réplica.

- El TDM en el nodo 1 le indica al TSA de la recuperación de su dispositivo.
- El TSA verifica si reconfiguró su réplica (si existe), en caso de ser así el TSA pone como réplica al dispositivo en el nodo 1 y como activo en el nodo (nodo 3) que lo contenía replicado, se actualiza y actualiza a todos los TSP.

c) Cuando el sistema se recuperó y no había una réplica o doble conexión del dispositivo múltiple en otro nodo y éste no era crítico.

- El TDM en el nodo 1 le indica al TSA de la recuperación de su dispositivo.
- El TSA al darse cuenta que no se reconfiguró al dispositivo en algún nodo le comunica al TSP en el nodo 1 que active al dispositivo y sus tareas.
- El TSP busca en su B.C. todas las tareas que dependen directamente o indirectamente del dispositivo, se comunica con los TFT de estas tareas informándoles que activen su tarea respectiva y le informa al TDM de la doble conexión del dispositivo que active a su dispositivo.
- El TDM y los TFT activan su correspondiente dispositivo o tarea y se lo comunican al TSP del nodo 1.
- El TSP 1 se actualiza y actualiza al TSA, el TSA se actualiza a todos los TSP.

Excepciones:

El TDM al momento de verificar si existen más dispositivos activos en el nodo analiza si con el número de dispositivos que quedan en el nodo pueden estos seguir activos en el sistema, sino, los da de baja y se lo comunica al TSA, el cual a su vez dependiendo del tipo de dispositivo puede o no cambiar el tipo de dispositivo en un nodo. Por ejemplo, si existen varios sensores de diferente tipo entre los nodos, por decir, 10 sensores infrarrojos y 10 bumpers, estos dispositivos serían del tipo no crítico redundante ya que ambos realizan de cierta manera una misma tarea (evitar obstáculos), al momento de fallar alguno de los dos nodos, el nodo activo pasaría a ser de tipo crítico redundante, crítico porque no existe otro tipo de sensores que puedan trabajar en la misma tarea y redundante porque existen varios sensores del mismo tipo en el nodo, lo cual hasta cierto grado permitiría seguir trabajando con la tarea.

Apéndice B

Caso de Meta: Tolerador de Fallos a Nivel Nodo en un Dispositivo Único

a) Objetivo: Tolerar los fallos del dispositivo único que conforman el nodo, ya sea actuador o sensor.
b) Precondiciones: El tolerador a fallos nivel nodo, nivel tarea y nivel sistema deben estar listos al iniciar el sistema. El sistema de control del robot este listo y correcto para trabajar.
c) Iniciador: El Tolerador a Fallos Nivel Nodo en Dispositivo Único comienza a monitorear al dispositivo, y el sistema de control debe estar corriendo.
d) Post-condiciones: El Tolerador a Fallos Nivel Nodo en Dispositivo Único detectó fallo en el dispositivo o está libre de fallos.
e) Roles:

- Tolerador de Fallos Nivel Nodo en Dispositivo Único (este rol será referido en el texto como TDU).
- Tolerador de Fallos Nivel Tarea (en el texto se referirá a este rol con el nombre de TFT).
- Tolerador de Fallos Nivel Sistema Activo (nos referiremos a este rol como TSA).
- Tolerador de Fallos Nivel Sistema Pasivo (nos referiremos a este rol con el nombre TSP).

f) Descripción del Rol:

1. Inicia cuando se activa el sistema de control del robot y todas las entradas de error están libre de error.
2. El TDU le informa al TSA que le dé de alta. El TSA lo registra, se actualiza y se lo comunica a todos los TSP.
3. El TDU inicia su monitoreo en el dispositivo, éste puede detectar o no un fallo en dicho dispositivo;
   a) Si el dispositivo único está libre de error continúa monitoreándolo.
   b) Si detecta fallo en el dispositivo pone al nodo en estado sospechoso, se lo comunica al TSA; el TSA se actualiza, envía un mensaje de actualización a todos los TSP; entra a la fase de localización.
4. En la fase de localización efectúa un test de corroboración de fallo al dispositivo; al realizar éste test al dispositivo puede obtener dos hechos:
   a) El dispositivo pasó la prueba: Al pasar el dispositivo la prueba de corroboración, el TDU continúa monitoreándolo, pone al nodo en estado recuperado, se lo comunica al TSA, el TSA se actualiza, envía un mensaje de actualización a todos los TSP.
   b) El dispositivo no pasó la prueba: Al no pasar un dispositivo la prueba de corroboración, el TDU lo pone en falla temporal, al nodo en estado degradado y pasa a la fase de asilamiento (paso 5) y a la fase de recuperación (paso 7); se lo comunica al TSA, el TSA se actualiza, envía un mensaje de actualización a todos los TSP y pasa a la fase de reconfiguración (paso6).
5. En la fase de aislamiento el TDU realiza las siguientes acciones:

   - El TDU realiza una serie de acciones para aislar el dispositivo único y se lo comunica a su TSP.
   - El TSP realiza una búsqueda de todas las tareas que dependen directamente o indirectamente del dispositivo que falló, se comunica con los TFT de estas tareas informándoles que se preparen para dormir su tarea respectiva, de la misma manera le informa al TDU que duerma su dispositivo.

- El TDU y los TFT duermen a su dispositivo o tarea respectivamente y se lo comunican a su TSP.
- El TSP se actualiza, actualiza al TSA, el TSA envía un mensaje de actualización a todos los TSP.
- Al finalizar el sistema continúa con las acciones realizadas en el punto 6.

6. En la fase de reconfiguración cuando el TDU le envía al TSA un mensaje de actualización en la fase de localización, el TSA al percatarse del fallo de el dispositivo realiza las siguientes acciones:

- El TSA busca en su B.C. el nodo que contiene la doble conexión de este dispositivo (supongamos se encuentra en el nodo 2) para cerciorarse de su fallo o del fallo de su conexión, igualmente busca el nodo que contiene la réplica del dispositivo (en caso de existir), pongamos como ejemplo que su réplica se encuentra en el nodo 3. Pueden suceder los siguientes casos:

a) El TSA envía un mensaje al TDU del nodo 2 que contiene la doble conexión del dispositivo que falló, en dicho mensaje le informa que pruebe si funciona el dispositivo.

- El TDU del nodo 2 realiza un procedimiento de verificación para cerciorarse del fallo del dispositivo.
- El TDU le indica al TSA el resultado del procedimiento; es decir si el dispositivo funciona o no funciona en ese nodo.
- En caso de funcionar el dispositivo, el TDU espera a que el TSP le informe si deberá activar al dispositivo o no; estas acciones se especifican en el punto 6.c.1. (esto se realiza con la finalidad de que el nodo que contiene la doble conexión no trabaje de manera degradada y que el TDU en el nodo 1 se entere si falló el dispositivo o la conexión del dispositivo).

b) Existe réplica de dicho dispositivo.

- El TSA le comunica al TSP del nodo 3 en el cual se encuentra la réplica del dispositivo que falló, que debe reconfigurar al dispositivo y sus tareas.
- El TSP busca en su B.C. todas las tareas que dependen directamente o indirectamente del dispositivo, se comunica con los TFT de estas tareas informándoles que activen su tarea respectiva y le informa al TDU del dispositivo replicado que active a su dispositivo.
- El TDU y los TFT activan al dispositivo y tarea respectivos.
- El TDU y los TFT le comunican al TSP del nodo 3 que han activado su dispositivo o tarea según sea el caso.
- El TSP se actualiza, actualiza al TSA, el TSA actualiza a todos los TSP.

c) No existe réplica de dicho dispositivo. En este caso puede acontecer lo siguiente:
1. Si el dispositivo funciona en el nodo 2, el cual contiene la doble conexión:

- El TSA al recibir el mensaje por parte del TDU del nodo 2 en el que le indica que el dispositivo funciona y al percatarse que no existe réplica de dicho dispositivo le comunica al TSP del nodo 2 el cual contiene la doble conexión del dispositivo, que debe reconfigurar al dispositivo y sus tareas.
- El TSP del nodo 2 busca en su B.C. todas las tareas que dependen directamente o indirectamente del dispositivo, se comunica con los TFT de estas tareas informándoles que activen su tarea respectiva y le informa al TDU de la doble conexión del dispositivo que active a su dispositivo.

- El TDU y los TFT activan su tarea y dispositivo respectivos.
- El TDU y los TFT le comunican a su TSP (TSP2) que han activado su dispositivo y tarea respectivamente.
- El TSP2 se actualiza, actualiza al TSA, el TSA actualiza a todos los TSP.

2. El dispositivo no funciona en el nodo 2.

- El TSA al recibir el mensaje por parte del TDU del nodo 2 en el que le indica que el dispositivo no funciona y al percatarse que no existe réplica de dicho dispositivo busca en su B.C. si existe otra doble conexión en un tercer nodo; si existe se realizan las mismas acciones que en el paso 6.a., en esta ocasión activa al dispositivo.

3. El dispositivo no funciona en su doble conexión y no existe réplica de dicho dispositivo.

- Al no funcionar el dispositivo en los nodos que contienen su doble conexión y al no contar con réplica de este dispositivo, el TSA espera un momento determinado a que el TDU en el nodo 1 le indique que ha recuperado al dispositivo, en este caso realiza las acciones definidas en el punto 7.b.3, si funciona lo activa.
- Si el TDU le indica al TSA que no ha recuperado el dispositivo en el nodo 1, el TSA realiza las acciones del punto 7.a.

7. Posterior a la recuperación se entra a la fase de recuperación donde se trata de recuperar al dispositivo fallido en la que pueden suceder los siguientes casos:

a) No se recupera el dispositivo.

- El TDU del nodo 1 le indica al TSA que ponga al dispositivo en baja permanente (no fue recuperado).
- El TSA al percatarse de esto, busca en su B.C. si dicho dispositivo es crítico o no es crítico.
- En caso de ser un dispositivo crítico el TSA realiza un paro seguro del sistema.
- En caso de no ser crítico el dispositivo el TSA continúa con las acciones tomadas en la fase de reconfiguración en caso de haberse activado el dispositivo en algún nodo; en caso contrario el TSA reconfigura al sistema sin dicho dispositivo.

b) Se recupera el dispositivo que falló.

1. Cuando el sistema se reconfiguró usando la doble conexión.

- El TDU en el nodo 1 le indica al TSA de la recuperación de su dispositivo.
- El TSA al percatarse de la recuperación del dispositivo le comunica le comunica al TSP en el nodo 2 (doble conexión) que duerma al dispositivo y sus tareas.
- El TSP del nodo 2 busca en su B.C. todas las tareas que dependen directamente o indirectamente del dispositivo, se comunica con los TFT de estas tareas informándoles que duerman su tarea respectiva y le informa al TDU que contiene la doble conexión del dispositivo que duerma la señal de su dispositivo.
- El TDU y los TFT duermen su respectivo dispositivo o tarea.
- EL TDU y los TFT le comunican al TSP del nodo 2 que han dormido su dispositivo y tarea respectivamente.
- El TSP se actualiza, actualiza al TSA.

- El TSA al recibir los mensajes por parte del TSP del nodo 2 que ha dormido respectivamente al dispositivo y tareas le comunica al TSP1 que active al dispositivo y tareas en el nodo 2.
- El TSP del nodo 1 busca en su B.C. todas las tareas que dependen directamente o indirectamente del dispositivo, se comunica con los TFT de estas tareas informándoles que activen su tarea respectiva y le informa al TDU del dispositivo que active a su dispositivo.
- El TDU y los TFT activan al dispositivo y las tareas respectivamente.
- EL TDU y los TFT le comunican al TSP del nodo 1 que han activado su dispositivo y tarea respectivamente.
- El TSP se actualiza, actualiza al TSA.

2. Cuando el sistema se reconfiguró usando la réplica.

   - El TDU en el nodo 1 le indica al TSA de la recuperación de su dispositivo.
   - El TSA verifica si reconfiguró su réplica (si existe), en caso de ser así el TSA pone como réplica al dispositivo en el nodo 1 y como activo en el nodo que lo contenía replicado (nodo 3).
   - El TSA se actualiza, actualiza a todos los TSP.

3. Cuando el sistema se recuperó y no había una réplica o doble conexión del dispositivo en otro nodo y éste no era crítico.

   - El TDU en el nodo 1 le indica al TSA de la recuperación de su dispositivo.
   - El TSA al darse cuenta de la recuperación del dispositivo, busca si dicho dispositivo se reconfiguró en algún nodo; al darse cuenta que no fue reconfigurado y que no era crítico le comunica al TSP en el nodo 1 que active al dispositivo y las tareas.
   - El TSP en el nodo 1 busca en su B.C. las tareas que dependen de dicho dispositivo y les informa que activen su respectiva tarea, igualmente le comunica al TDU del dispositivo que debe activar al dispositivo.
   - El TDU y los TFT activan su dispositivo y tarea respectivos, le comunican de ello a su TSP.
   - El TSP se actualiza y actualiza al TSA. El TSA actualiza a todos los TSP.

Apéndice C

A continuación se propone un Shell genérico, para poder implementar cualquier mecanismo tolerante a fallos en los agentes, y se describen las 3 Bases de Conocimiento para cada agente que integra el SMA tolerante a fallos propuesto en este trabajo de investigación .

Agente Nodo (AN_i)

Base de Conocimiento del Agente Nodo

DETECION

Si
(AN_i).Fase.Detección **y**
(AN_i).{Entrada-Error ($_{i,j}$).Error}
Entonces
(AN_i).Estado.Sospechoso **y**
(AN_i).Fase.Localización

LOCALIZACION

Si
(AN_i).Fase.Localización **y**
(AN_i).{Entrada-Error($_{i,j}$).Error} **y**
(AN_i).Estado.Sospechoso
Entonces
(AN_i).Efectuar- test-a-los-diferentes-dispositivos-del (N_i)
Si
(AN_i).Fase.Localización **y**
(AN_i).Entrada-Error($_{i,j}$).Error **y**
(AN_i).Test [$D_{i,m}$].No-Pasa
Entonces
(AN_i).Dispositivo[$D_{i,m}$].Incorrecto **y**
(AN_i).Estado.Degradado
Si
(AN_i).Fase.Localización **y**
(AN_i).Dispositivo[$D_{i,m}$].Incorrecto **y**
(AN_i).Estado.Degradado
Entonces
(AN_i).Acciones-para-detectar-tipo-Dispositivo($_m$)
Si
(AN_i).Fase.Localización **y**
(AN_i).Dispositivo[$D_{i,m}$].Incorrecto **y**
(AN_i).Estado.Degradado **y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$].No-Crítico *(Fallo un sensor y/o actuador redundante)*
Entonces
(AN_i).Dispositivo[$D_{i,m}$].Baja **y**
($AN_i AT_j$).Estado.A-Reconfigurar($_m$) **y**
($AN_i AS$).Estado-Degradado **y**
($AN_i AT_j$).Estado-Dispositivo[$D_{i,m}$].Baja **y**
($AN_i AS$).Estado-Dispositivo[$D_{i,m}$].Baja **y**
(AN_i).Fase.Aislamiento

Si
(AN_i).Fase.Localización **y**
(AN_i).Dispositivo[$D_{i,m}$].Incorrecto **y**
(AN_i).Estado.Degradado **y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$].S-Crítico *(Fallo un sensor y/o actuador no redundante)*
Entonces
(AN_i).Dispositivo[$D_{i,m}$].Baja **y**
(AN_i*AS*).Estado.Degradado **y**
(AN_i*AT*$_j$).Estado-Dispositivo[$D_{i,m}$].Baja **y**
(AN_i*AS*).Estado-Dispositivo[$D_{i,m}$].Baja **y**
(AN_i).Fase.Aislamiento
Si
(AN_i).Fase.Localización **y**
(AN_i).Dispositivo[$D_{i,m}$].Incorrecto **y**
(AN_i).Estado.Degradado **y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$].Crítico-Redundante
Entonces
(AN_i).Dispositivo[$D_{i,m}$].Baja **y**
(AN_i*AS*).Estado-Dispositivo[$D_{i,m}$].Baja **y**
(AN_i).Fase.Aislamiento

AISLAMIENTO

Si
(AN_i).Fase.Aislamiento **y**
(AN_i).Dispositivo[$D_{i,m}$].Baja **y**
(AN_i).Estado.Degradado
Entonces
(AN_i).Acciones-Aislamiento-Dispositivo($_m$) **y**
(AN_i).Fase.Reconfiguración

RECONFIGURACION
Si
(AN_i).Fase.Reconfiguración **y**
(AN_i).Estado.Degradado **y**
(AN_i).Dispositivo[$D_{i,m}$].Baja
Entonces
(AN_i).Acciones-Reconfiguración-Dispositivo($_m$) **y**
(AN_i).Fase.Recuperación

RECUPERACION
Si
(AN_i).Fase.Recuperación **y**
(AN_i).Dispositivo[$D_{i,m}$].Baja **y**
[(AT_jAN_i).Estado.Reconfigurada**] o [**(AT_jAN_i).Estado.No-Reconfigurada**] y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$].No-Crítico
Entonces
(AN_i).Checa-Si-existen-mas dispositivos-de-e/s-Activos(N_i)
Si
(AN_i).Fase.Recuperación **y**
(AT_jAN_i).Estado.Reconfigurada **y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$].No-Crítico **y**
(AN_i).Si-Existen-mas-dispositivos-de-e/s-activos(N_i).
Entonces

(AN_i).Fase.Detección
Si
(AN_i).Fase.Recuperación **y**
(AT_jAN_i).Estado.No-Reconfigurada **y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$].No-Crítico **y**
(AN_i).Si-Existen-mas-dispositivos-de-e/s-activos(N_i).
Entonces
(AN_i).Fase.Detección **y**
(AN_i).Todos-Dispositivo-con-fallos-Mismo-Tipo(z).Baja **y**
($AN_i AS$).Estado.Todos-Dispositivos-Mismo-Tipo(z).Baja
Si
(AN_i).Fase.Recuperación **y**
(AT_jAN_i).Estado.No-Reconfigurada **y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$].No-Crítico **y**
(AN_i).No-Existen-mas-dispositivos-de-e/s-activos(N_i).
Entonces
(AN_i).Estado.Baja **y**
(AN_iAS).Estado.Baja **y**
(AN_iAT_j).Estado.Baja
Si
(AN_i).Fase.Recuperación **y**
(AN_i).Estado.Degradado **y**
(AN_i).Dispositivo[$D_{i,m}$].Baja **y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$].S-Crítico
Entonces
(AN_i).Checa-Si-existen-mas dispositivos-de-e/s-Activos(N_i)
Si
(AN_i).Fase.Recuperación **y**
(AN_i).Estado.Degradado **y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$].S-Crítico **y**
(AN_i).Dispositivo[$D_{i,m}$].Baja **y**
(AN_i).Si-Existen-mas-dispositivos-de-e/s-activos(N_i).
Entonces
(AN_i).Fase.Detección
Si
(AN_i).Fase.Recuperación **y**
(AN_i).Estado.Degradado **y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$].S-Crítico **y**
(AN_i).Dispositivo[$D_{i,m}$].Baja **y**
(AN_i).No-Existen-mas-dispositivos-de-e/s-activos(N_i).
Entonces
(AN_i).Estado.Baja **y**
(AN_iAS).Estado.Baja **y**
(AN_iAT_j).Estado.Baja
Si
(AN_i).Fase.Recuperación **y**
(AN_i).Estado.Degradado **y**
(AN_i).Dispositivo[$D_{i,m}$].Baja **y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$].Crítico-Redundante
Entonces
(AN_i).Checa-Si-existen-mas dispositivos-de-e/s-Activos(N_i)
Si
(AN_i).Fase.Recuperación **y**
(AN_i).Estado.Degradado **y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$]. Crítico-Redundante **y**

(AN_i).Dispositivo[$D_{i,m}$].Baja **y**
(AN_i).Si-Existen-mas-dispositivos-de-e/s-activos(N_i).
Entonces
(AN_i).Fase.Detección
Si
(AN_i).Fase.Recuperación **y**
(AN_i).Estado.Degradado **y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$]. Crítico-Redundante **y**
(AN_i).Dispositivo[$D_{i,m}$].Baja **y**
(AN_i).No-Existen-mas-dispositivos-de-e/s-activos(N_i).
Entonces
(AN_i).Estado.Baja **y**
(AN_i*AS*).Estado.Baja **y**
($AN_i$$AT_j$*)*.Estado.Baja

Agente Tarea (AT_j)

Base de Conocimiento del Agente Tarea

DETECCION

Si
(AT_j).Fase.Detección **y**
(AT_j).{Entrada-Error.Error} **y**
(AN_i*AT_j*).Estado-Dispositivo[$D_{i,m}$].Baja **y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$].S-Crítico (sensor y/o actuador no redundante fallo en N_i)
Entonces
(AT_j).Estado.Baja **y**
(AT_j*AS*).Estado.Baja **y**
(AT_j*AT_k*).Estado.Baja **y**
(AT_j).Fase.Aislamiento
Si
(AT_j).Fase.Detección **y**
(AT_j).{Entrada-Error.Error} **y**
(AT_k*AT_j*).Estado.Baja (Si la tarea de mas bajo nivel T_k falla y T_j trata de sobrevivir)
Entonces
(AT_j).Estado.Sospechoso **y**
(AT_j).Fase.Reconfiguración
Si
(AT_j).Fase.Detección **y**
(AN_i).Tipo-Dispositivo[$D_{i,m}$].No-Crítico **y**
($AN_i$$AT_j$)A-Reconfigurar($_m$) (Si un sensor o actuador redundante fallaron en el N_i)
Entonces
(AT_j).Fase.Reconfiguración

Si
(ATC_j).Fase.Detección **y**
(ATC_j).Entrada-Error.Error(Si falla un dispositivo crítico no redundanteb en N_i)
Entonces
(ATC_j).Estado.activada y
(ATC_j*AS*).Estado.activada y
(ATC_j*AN_k*).Estado.activada y
(ATC_j).Fase.reconfiguración

ASILAMIENTO

Si
(AT_j).Fase.Aislamiento **y**
(AT_j).Estado.Baja
Entonces
(AT_j).Acciones-de-aislamiento-para (T_j) **y**
(AT_j).Fase.Reconfiguración

RECONFIGURACIÓN

Si
(AT_j).Fase.Reconfiguración **y**
(AT_j).Estado.Sospechoso
Entonces
(AT_j).Acciones-de-reconfiguración-con-otras-tareas
(AT_j).Fase.Recuperación
Si
(AT_j).Fase.Reconfiguración **y**
(AT_j).Estado.Sospechoso
(AT_j).Sigue-Activa.Activa
Entonces
(AN_i).Fase.Recuperación
Si
(AT_j).Fase.Reconfiguración **y**
(AT_j).Estado.Sospechoso
(AT_j).Sigue-Activa. No-Activa
Entonces
(AT_j).Estado.Baja
(AT_j*AS*).Estado.Baja
Si
(ATC_j).Fase.Reconfiguración
(ATC_j).Estado.Activada
Entonces
(ATC_j).Acciones-de-reconfiguración-en-el-Nodo($_k$)

RECUPERACION

Si
(AT_j).Fase.Recuperación **y**
(AT_j).Sigue-Activa.Activa
Entonces
(AS).Acciones-Recuperacion-con-las-Tareas-Restantes
(AT_j).Fase.Detección
Si
(AT_j).Fase.Recuperación **y**
(AT_j).Sigue-Activa.No-Activa
Entonces
(AST_j).Estado.Baja

Agente Sistema (AS)

Base de Conocimiento del Agente Sistema

DETECCION

Si
(AS).Fase.Detección **y**
(AS){Entrada-Error.Error($_{Nodo}$)}
Entonces
(AS).Estado.Sospechoso($_{Nodo}$) **y**
(AS).Fase.Localización
Si
(AS).Fase.Detección **y**
(AS).{Entrada-Error.Error($_{Tarea}$)}
Entonces
(AS).Estado.Sospechoso($_{Tarea}$) **y**
(AS).Fase.Localización

LOCALIZACION

Si
(AS).Fase.Localización **y**
(AS). Estado.Sospechoso($_{Nodo}$) **y**
(AS).{Entrada-Error.Error($_{Nodo}$)}
Entonces
(AS).Efectuar- Test-a-los-diferentes-Nodos-del-Sistema
Si
(AS).Fase.Localización **y**
(AS).{Entrada-Error.Error($_{Tarea}$)} **y**
(AS).Estado.Sospechoso($_{Tarea}$)
Entonces
(AS).Efectuar- Test-a-las-diferentes-Tareas-del-Sistema
Si
(AS).Fase.Localización **y**
(AS).Entrada-Error.Error($_{Nodoi}$) **y**
(AS).Estado.Sospechoso($_{Nodoi}$) **y**
(AS).Test[$_{Nodoi}$]. No-Pasa
Entonces
(AS).Estado.Baja($_{Nodoi}$) **y**
(AS).Estado.Degradado($_{Nodoi}$) **y**
(AS).Fase Aislamiento-Nodo(i)
Si
(AS).Fase.Localización **y**
(AS).Entrada-Error.Error($_{Tareja}$) **y**
(AS).Estado.Sospechoso($_{Tareaj}$) **y**
(AS).Test($_{Tareaj}$). No-Pasa
Entonces
(AS).Estado.Baja($_{Tareaj}$) **y**
(AS).Estado.Degradado($_{Tareaj}$) **y**
(AS).Fase Aislamiento-Tarea($_{j}$)

AISLAMIENTO

Si
(AS).Fase Aislamiento-Nodo($_{i}$) **y**
(AS).Estado.Baja($_{Nodoi}$)
(AS).Estado.Degradado($_{Nodoi}$)

Entonces
(AS).Acciones-de-aislamiento-para-el Nodo($_i$) **y**
(AS).Fase.Reconfiguración-Nodo($_i$)
Si
(AS).Fase.Aislamiento-Tarea($_j$) **y**
(AS).Estado.Baja($_{tareaj}$)
(AS).Estado.Degradado($_{tareaj}$)
Entonces
(AS).Acciones-de-aislamiento-para-la-Tarea($_j$) **y**
(AS).Fase.Reconfiguración-Tarea($_i$)

RECONFIGURACION

Si

(AS).Fase.Reconfiguración-Nodo($_i$) y
(AS).Estado.Baja($_{Nodoi}$).Baja

(AS).Estado.Degradado($_{Nodoi)}$
Entonces
(AS).Acciones-de-Reconfiguración-con-los-Nodos-restantes
(AS).Fase.Recuperación-Nodo($_i$)

Si

(AS).Fase.Reconfiguración-Tarea($_j$) y
(AS).Estado.Baja($_{Tareaj}$)
(AS).Estado.Degradado($_{Tareaj}$)
Entonces
(AS).Acciones-de-reconfiguración-con-las-Tareas-restantes
(AS).Fase.Recuperación-Tarea($_j$)

RECUPERACION

Si

[(AS).Fase.Recuperación-Nodo($_i$)] o [(AS).Fase.Recuperación-Tarea($_j$)]

(AS).Estado.Degradado
Entonces
(AS).Acciones-de-recuperación **y**
(AS).Fase.Detección

Apéndice D

Desarrollo del Simulador

El simulador de la arquitectura tolerante a fallos, está compuesta por varias clases de agentes. En este caso se desarrollaron varios tipos de agentes:

- Agentes que toleran los fallos a nivel hardware, que se llaman agente nodo.
- Agentes que toleran los fallos a nivel software, que se llaman agente tarea y agente sistema.

Se hace uso de un agente nodo por cada dispositivo físico como lo son sensores o actuadores, microcontrolador y el controlador de la red. Un agente tarea por cada actividad que exista en el sistema de control, como control de trayectoria, el control de los motores, entre otras. Y un agente sistema que es el encargado del registro y control de los agentes.

Las clases fueron desarrolladas en el lenguaje de programación JAVA, ya que la herramienta de manejo JADE es el lenguaje que soporta. Todas las clases extienden de la clase Agent, esto para que puedan contar con ciertas características de agente y el manejador las pueda utilizar.

Dentro del cuerpo de los agentes, están los comportamientos que el agente realiza. De igual manera todos los agentes desarrollados cuentan con el manejo de mensajes para lograr una comunicación entre ellos, lo cual hizo posible el funcionamiento de esta arquitectura.

A continuación se describe cada una de las clases de los agentes implementados en el simulador.

- **Agente Sistema Activo.-** Este agente es el encargado en este caso de crear a los demás agentes que serán usados en un nodo, o host. Estos son el agente sensor, agente nodo, y agentes tareas. Una vez creados los agentes este los registra con el AMS para que la herramienta de manejo JADE los reconozca, y les otorgue un nombre y dirección. Ya creados y registrados, los agentes comienzan a realizar sus acciones. Este también envía continuamente mensajes al Agente Sistema pasivo Para conocer el estado de su funcionamiento. De la misma manera envía mensajes de activación a los agentes nodo y agentes tarea, para que estos despierten y activen a sus dispositivos o tareas. La tabla D.1 es un resumen del funcionamiento del Agente Sistema Activo, de la ejecución de las tareas que realiza y la comunicación que sostiene con los otros tipos de agentes.

Función del Agente Sistema Activo	*Ejecución de tareas*	*Mensajes*
Recibe mensaje de registro del Agente Nodo (AN).		***RegistrodeAN(ND,Nodo)***
Le asigna identificador único.	***AsignarIDalAN()***	
Le asigna el estado activo (E).	***AsignarEstadoalAN()***	
Lo anota en la bitácora de de registro, en que nodo se localiza (Nodo), estado(E), identificador único (ID), y dispositivo al que monitorea (ND)	***RegistrarAN(ID,E,ND,Nodo)***	
Recibe mensaje de registro del Agente Tarea (AT).		***RegistrodeAT(NT,nodo)***
Le asigna identificador único.	***AsignarIDAT()***	
Le asigna el estado activo (E).	***AsignarEstadoAT()***	
Lo anota en la bitácora de de registro, en que nodo se localiza (Nodo),	***RegistrarAT(ID,E,NT,Nodo)***	

estado(E), identificador único (ID), y tarea a la que monitorea (NT)		
Al terminar el registro de todos los agentes, envía el mensaje a todos los ASP para que actualicen su bitácora de registro		***ActualizacionRegistroAgentes*()**
Envía mensaje a todos los ASP para confirmar su funcionalidad, y recibe mensaje de funcionalidad de cada ASP		
Enviar mensaje continuamente a todos los ASP con el propósito de comunicarles que está activo(funcionando el nodo donde se localiza)		***EstoyActivoASA()***
Recibe continuamente mensaje de cada ASP con la finalidad de comunicarle que está activo(funcionando el nodo donde se localiza)		***NodoActivo(ID)***
Recibe y envía mensajes del Agente Tarea(AT) y el usuario		
Para que el usuario programe otra tarea ya que la tarea copia esta activa, en fallo, no tiene otra copia y es crítica		***ProgramacionTareaNuevaPorFallo TareaCriticoSinCopia(TC)***
Envía mensaje al usuario **que genere una tarea sin fallo y su copia, borre las que están en fallo, las active y actualiza al ASA y todos loa ASP**		***GerneraYActivaTareaYCopia(ID)***
Recibe mensaje del usuario **tarea y copia activada, funcionando y actualizada bitácora de fallos del ASA y todos los ASP**		***TareaYCopiaGeneradaYActivada(ID)***
Recibe y envía mensajes del Agente Nodo(AN) y del usuario		
Para que realice un paro seguro ya que la el dispositivo múltiple **en fallo, no tiene réplica y es crítico**		***ParoSeguroPorFallo DispCriticoMulSin Replica(ID)***
Envía mensaje al usuario que realizará un paro de manera segura al sistema de control.		***ParoSeguroPorFallo DispCriticoMulSin Replica(ID)***
Recibe confirmación del usuario		***OkRealizaElParo()***
Realiza la tarea para anotar en la bitácora de fallos el dispositivo que provoco el paro y el momento que falló, guarda la información que tenga el ASA en memoria y en su cola de mensajes y hace el paro seguro del sistema	***HacerParoSeguro(),***	
Para que realice la búsqueda de la réplica de un dispositivo múltiple y reconfigure el sistema al activarlo		***BuscarReplica DispMul(ID)***
Ejecuta la tarea para buscar el dispositivo duplicado en otro nodo	***BuscandoDispReplicadoEnOtroNodo()***	
Envía mensaje al AN del nodo donde se localiza duplicado el dispositivo para que los active y realice su reconfiguración en el nodo		***ActivarDispositivoDuplicado().***
Recibe mensaje del AN que actualice la bitácora de fallos		***ActualizaBitacoraFalloDDuplicado(ID)***
Ejecuta tarea de actualización de su	***ActualizaBitacoraFalloD***	

bitácora de fallos	***Duplicado(ID)***	
Envía mensaje de actualización de bitácora de fallos a todos los ASP.		***ActualizaBitacoraFalloD Duplicado(ID).***
Para que realice un paro seguro ya que la el *dispositivo único* **en fallo, no tiene réplica y es crítico**		***ParoSeguroPorFalloDisp CriticoUnicoSin Replica(ID).***
Envía mensaje al usuario que realizará un paro de manera segura al sistema de control del invernadero		***ParoSeguroPorFallo DispCritioUnicoSin Replica(ID)***
Recibe confirmación de paro del usuario		***OkRealizaParo()***
Ejecuta la tarea de hacer el paro, guarda la información que el ASA tenga en memoria y la información del contexto	***HacerParoSeguro()***	
Para que realice la búsqueda d e la réplica de un dispositivo único y reconfigure el sistema al activarlo		***BuscaLaReplicaDispUni co(ID)***
Ejecuta la tarea para buscar el nodo donde se localiza la réplica del dispositivo único	***BuscandoDispReplicado EnOtroNodo()***	
Envía mensaje al AN del nodo donde se localiza duplicado el dispositivo para que los active y realice su reconfiguración en el nodo		***ActivarDispositivoDuplic ado()***
Recibe mensaje del AN que actualice la bitácora de fallos		***ActualizaBitacoraFalloD Duplicado(ID)***
Ejecuta tarea de actualización de su bitácora de fallos	***ActualizaBitacoraFalloD Duplicado(ID)***	
Envía mensaje a todos lo ASP para que actualicen la bitácora de fallos		***ActualizaBitacoraFalloD Duplicado(ID)***
Envía mensaje a un Agente nodo que está dormido		
Envía mensaje de activación a un Agente Nodo dormido con el propósito que active a un dispositivo duplicado		***ActivarANDeDispositivo Duplicado(ID)***
Recibe mensaje de actualizar bitácora de fallos por activación de dispositivo duplicado		***Esta ActivadoDispositivo Duplicado(ID)***
Actualiza Bitácora de fallos	***ActualizarBitácora DeFallosDispDuplicado(ID)***	
Envía mensaje actualización de bitácora de fallos por activación de dispositivo duplicado a todos los ASP		***ActualizarBitacora FallosDispositivoDuplica do(ID)***

Tabla D.1 Actividades realizadas por el Agente Sistema Activo.

- ***Agente Sensor.-*** Este agente sensor se desarrolló con la finalidad de simular el funcionamiento de un sensor, y el envío de mensajes al agente al agente nodo que lo controla con mayor facilidad. Dicho agente al momento de ser registrado inicia su actividad, enviando mensajes con valores al agente nodo. Al momento de enviar erróneamente un valor recibe un mensaje del agente nodo, este detiene su funcionamiento con la finalidad de no enviar valores erróneos por más tiempo.

- ***Agente Nodo.-*** Dicho agente fue desarrollado para monitorear a un dispositivo físico ya sea sensor ó actuador. Recibe constantemente señal del dispositivo, y en caso de que

falle, el agente nodo lo detecta y lo aísla., y envía un mensaje al Agente Sistema indicando que el dispositivo falló. De igual manera si el dispositivo tiene relación con alguna tarea, el Agente Nodo le comunica al agente tarea que que pare, ya que el valor que evia es erróneo. El Agente Nodo puede controlar dispositivos únicos o múltiples. Éste puede estar dormido en otro nodo y recibir mensajes del agente sistema, con la finalidad de que active a un dispositivo replicado, ya que el dispositivo en cuestión ha fallado en otro nodo. La tabla D.2 es un resumen del funcionamiento del Agente Nodo, de la ejecución de las tareas que realiza y la comunicación que sostiene con los otros tipos de agentes.

Función del Agente Nodo	*Ejecución de tareas*	*Mensajes*	*DE A*
Se Registra ante el ASA al enviar mensaje.		***RegistrodeAN(ID)***	ASA
Monitoreo continúo del dispositivo a su cargo propósito detectar fallo. Detección de fallo en un dispositivo múltiple	***MonitorearDispositivo(ID).***		
Checa rango o umbral. *Caso 1 está dentro de rango*	***VerificarDispMul(ID)***		
Descuenta elemento en fallo.	***DescontarUnSoloDispPorFallo(ID)***		
Pasar parámetro de descuento a la tarea *VerificarDispMul(ID*).	***PasoDeParametro(ID,N)***		
Desactiva elemento en fallo.	***DesactivarSoloUnElemntoDelDispMulEnFallo(ID)***		
Envía mensaje para actualización de bitácora de fallos, inicia recuperación del elemento en fallo, *fin caso 1.*		***ActualizaBitacoraFalloParcialDM(ID)***	ASP
Caso 2 está fuera de rango			
Descuenta todo dispositivo múltiple.	***DescontarTodoDispMulPorFallo(ID)***		
Pasar parámetro de descuento a la tarea *VerificarDispMul(ID*).	***PasoDeParametro(ID,N)***		
Desactiva el dispositivo múltiple.	***DesactivarTodoDispMulEn Fallo(ID)***		
Envía mensaje interno para actualización de bitácora de fallos.		***ActualizaBitacora FalloTotalDM(ID)***	ASP
Desactivado el dispositivo busca si es crítico y sin réplica.	***BuscarReplicaCritico DispMul(ID)***		
Si el resultado de la tarea *BuscarReplicaCriticoDispMul(ID)* es positivo envía mensaje para realizar un paro seguro.		***ParoSeguroPorFallo DispCriticoMulSin Replica(ID)***	ASA
Si el resultado de la tarea *BuscarReplicaCriticoDispMul(ID)* es negativo busca si tiene réplica	***BuscarReplicaDisp Mul(ID)***		
Si el resultado de la tarea *BuscarReplicaDispMul(ID)*es positivo envía mensaje, con la finalidad de que el ASA reconfigure el sistema de control al entrar en funcionamiento la réplica.		***BuscarReplica DispMul(ID)***	ASA
Si el resultado de la tarea *BuscarReplicaDispMul(ID)* es negativo			

inicia recuperación. *Fin del caso 2.* Detección de fallo en un dispositivo único.			
Desactiva el dispositivo en fallo	***DesactivarDispUnicoEnFallo(ID)***		
Enviar mensaje de actualización de bitácora de fallos al ASP		***ActualizaBitacoraFalloDispUnico(ID)***	ASP
Buscar si no tiene réplica y es crítico	***BuscarReplicaCriticoDispUnico(ID)***		
Si el resultado de *BuscarReplicaCriticoDisp*			
***Unico(ID),* es positivo envía mensaje al ASA para que realice un paro seguro**		***ParoSeguroPorFalloDisp CriticoUnicoSin Replica(ID).***	ASA
Si el resultado de *BuscarReplicaCriticoDisp Unico(ID*) es negativo busca si tiene réplica	***BuscarReplicaDispUnico(ID)***		
Envía un mensaje al ASA para que busque la réplica del dispositivo único y reconfigure el sistema al poner la replica activa		***BuscaLaReplicaDispUnico(ID)***	ASA
Recibe mensajes y el agente nodo está "dormido"			
Caso 1 **Recibe mensaje para activar al Agente nodo, con el objetivo de que éste active al dispositivo replicado y tolere sus fallos**		***ActivarANDeDispositivo Duplicado(ID*)**	ASA
Activación del agente Nodo	***ActivarAgenteNodoDormido(ID)***		
El AN activa al dispositivo replicado	***ActivarDispositivoReplicado(ID)***		
Envía mensaje de activación a los agentes tareas que tienen relación con el dispositivo con el objetivo que estos las activen y toleren sus fallos.		***ActivarAgenteTareaPor DispDuplicado(ID*)**	AT
Envía mensaje para actualizar bitácora de fallos.		***EstaActivadoDispositivoPorReplica(ID*)**	ASA

Tabla D.2 Actividades realizadas por el Agente Nodo.

- **Agente Tarea.-** eEs el encargado de monitorear a las tareas que ntegran al l sistema de control. En este caso se usaron tareas relativamente simples, de tal manera que el simulador pueda cumplir su cumpla la tolerancia a fallos. El agente tarea monitorea las tareas de control, y al fallar una de ellas, dicho agente la desactiva y activa su copia. El Agente Tarea le informa al Agente Sistema que la tarea falló, y que se activó su copia. Al fallar la copia, el agente tarea le informa al Agente Sistema para que mande un mensaje de alerta al usuario, de tal manera que este carge al sistema otra tarea idéntica al que fallo. Cuando se trata de un Agente Tarea replicado, en estado dormido al recibir el mensaje de activación del Agente Sistema, el mencionado Agente Tarea se activa y activa las tareas que estan a su cargo. La tabla D.3 es un resumen del funcionamiento del Agente Tarea, de la ejecución de las tareas que realiza y la comunicación que sostiene con los otros tipos de agentes.

Funciones del Agente Tarea	Ejecución de tareas	Mensajes enviados o recibidos	A DE
Se Registra ante el ASA al enviar el mensaje		***RregistrodeAT(ID)***	ASA
Monitoreo continuamente de la tarea a su cargo, finalidad detectar fallo	***MonitorearTarea(ID).***		
Si detecta el fallo envía mensaje		***PonerEnFalloTarea(ID)***	ASP o ASA
Desactiva tarea en fallo (aisla)	***DesactivarTarea(ID)***		
Activa tarea copia	***ActivaTareaCopia(ID)***		
Buscar AN por si existe relación con dispositivo de E/S, para direccional señal	***BuscarRelAgenteNodo(ID)***		
Si el resultado de la tarea *BuscarAgenteNodo(ID)* es positivo, envía mensaje al AN		***DireccionarDispositivoATarea Copia(ID)***	AN
Recibe mensaje de confirmación de direccionamiento del AN		***RealizadoDireccionamiento ALaTareaCopia(ID)***	AN
Buscar AT por si existe relación con otras tareas, para direccional datos.	***BuscarRelAgenteTarea(ID)***		
Si el resultado de la tarea *BuscarAgenteTarea(ID)* es positivo, envía mensaje al AT		***DireccionalTareaATarea Copia(ID)***	AT
Recibe mensaje de confirmación de direccionamiento del AT		***RealizadoDireccionamiento AlaTareaCopia(ID)***	AT
Enviar mensaje al ASP de actualización de bitácora		***ActualizarBitacoraFallos PorTarea(ID)***	ASP o ASA
Si existe un fallo en la copia la desactiva.	***DesactivarTarea Copia(ID)***		
Busca si es critica y no tiene otra tarea copia	***ChecarTareaCopiaCrítica(ID)***		
Si el resultado de la tarea *ChecarTareaCopiaCritica(ID)* es positivo, envía al usuario un mensaje para que programe otra tarea.	**.**	***ProgramacionTareaNueva PorFalloTarea CriticoSinCopia(TC).***	USUARIO
En estado dormido puede recibir mensajes			
Checar cola de mensajes, en caso de recibir mensaje de activación del ASP ó del ASA	***ChecarColaDeMensajes(ID).***		
Si recibe mensaje de activación por réplica de dispositivo		***ActivarAgenteTareaPorDispD uplicado(ID)***	ASP
Se Activa al Agente tarea	***ActivarAgenteTarea(ID)***		
Recibido el mensaje de activación, el AT ya activado , activa a la tarea correspondiente	***ActivaTareaPorReplica(ID)***		
Envía mensaje a su ASP de que se activo el AT y su tarea , con el propósito de actualizar la bitácora de fallos		***EstaActivadaTareaPorReplica (ID)***	ASP

Tabla D.3 Actividades realizadas por el Agente Tarea.

- **Agente sistema Pasivo.-** Es una copia al Agente Sistema Activo, su funcionamiento es controlar a los agentes que se encuentran en su nodo, recibe continuamente mensajes del Agente Sistema Activo. De tal manera que si un dispositivo del nodo donde se encuentra el Agente Sistema Activo llegase a fallar, el Agente Sistema Pasivo activa su réplica en el otro nodo de tal manera que el sistema de control prosigue su funcionamiento. Al fallar un dispositivo que es crítico y no tiene réplica, este manda un paro seguro del sistema de control. . La tabla D.4es un resumen del funcionamiento del Agente Sistema Activo, de la ejecución de las tareas que realiza y la comunicación que sostiene con los otros tipos de agentes

Función del Agente Sistema Pasivo	Ejecución de tareas	Mensajes	DE A
Se Registra ante el ASA al enviar mensaje.		***RegistrodeASP(LD,ID)***	ASA
Recibe mensaje de actualizar bitácora de registro		***ActualizacionRegistroAgentes()***	ASA
Envía continuamente al ASA mensaje de que está activo		***NodoActivo(ID)***	ASA
Recibe continuamente mensaje de que está activo el ASA		***EstoyActivoASA()***	ASA
Recibe el mensaje del AT para poner en fallo la tarea		***PonerEnFalloTarea(ID)***	AT
Desactiva la tarea que fallo	***DesactivarTareaEnFallo(ID)***		
Envía mensaje de activación del Agente tarea para que este active las tareas debido a que se requiere activar un dispositivo replicado en el nodo donde el AT está dormido		***ActivarAgenteTareaPorDispDuplicado(ID)***	AT
Recibe mensaje del su AT que activo al agente tarea por fallo en nodo y se requiere activar su réplica		***EstaActivadaTareaPorReplica(ID)***	AT

Tabla D.4 Actividades realizadas por el Agente Sistema Pasivo.

Prueba fallo de una tarea en el simulador

La primera prueba es cuando falla una tarea. El agente tarea se encuentra monitoreando a su tarea correspondiente, en la figura D.1 se muestran las pantallas del simulador funcionando con la tarea funcionando correctamente en el Nodo1, el Nodo2 esta siendo monitoreado continuamente y en espera de ser llamado.

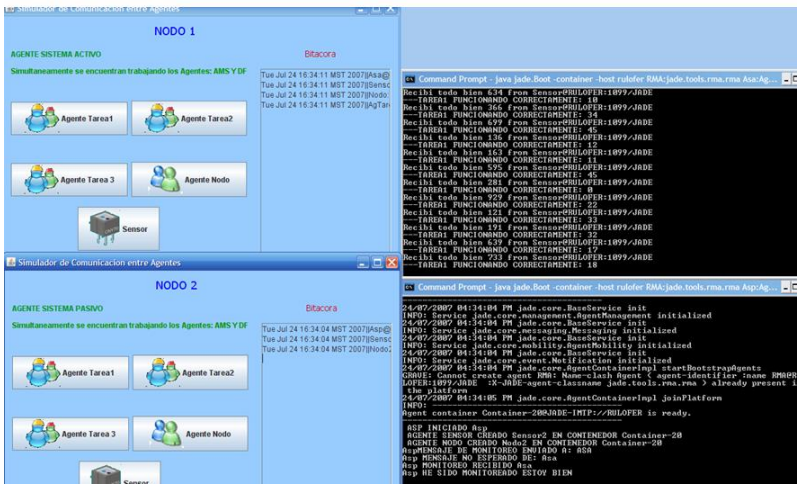


Figura D.1 Tarea funcionando correctamente.

Cuando la tarea falla el Agente Tarea detecta al fallo desactiva a la tarea, envia un mensaje de redireccionamiento de señal al Agente Nodo y activa la tarea copia. En la figura D.2 se muestra como actúa el Agente Tarea cuando falla una tarea en el Nodo1, el Agente Tarea activa a la tarea copia. El Nodo2 sigue en espera de ser llamado para activar a sus agentes.

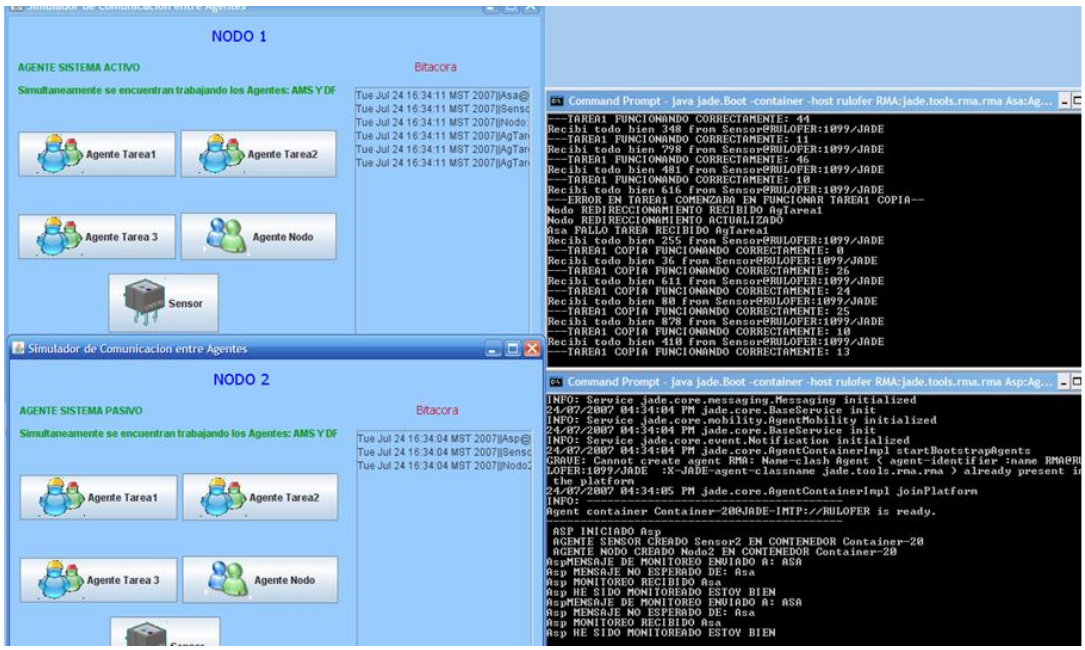


Figura D.2 Fallo tarea y activación de tarea copia.

Si la tarea copia falla, el Agente Tarea informa al Agente Sistema Activo para que envie un mensaje al usuario y éste de de alta una tarea idéntica a la que falló. En la figura D.3 se muestra como aparece un mensaje de advertencia al usuario al momento de fallar la tarea copia.

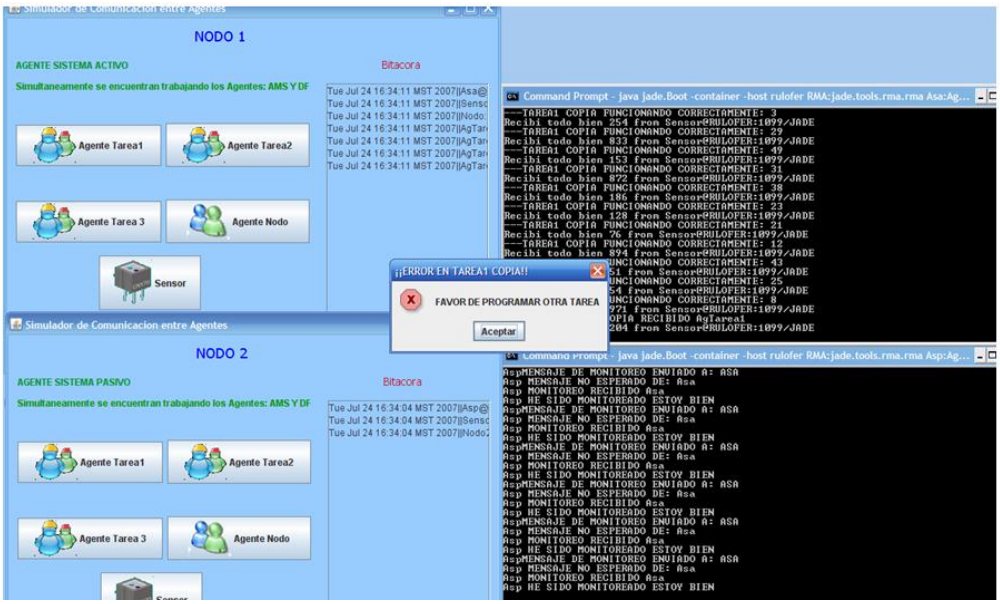


Figura D.3 Fallo tarea copia y mensaje al usuario

Cuando la tarea en fallo se desactiva hasta que el usuario carge una nueva tarea. La figura D.4 muestra como el Agente Tarea desactiva a la tarea en fallo.

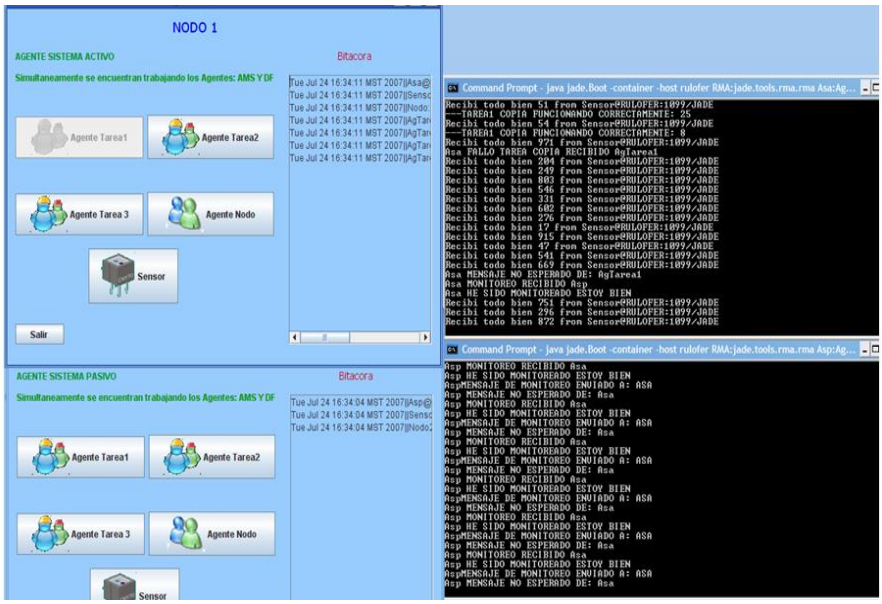


Figura D.4 Desactivación de la tarea en fallo.

Prueba fallo de dispositivo en el simulador

La segunda prueba es la del fallo de un dispositivo, en la figura D.5 se puede apreciar que en el Nodo1 recibe correctamente los valores del sensor. El Nodo2 se encuentra en espera de un mensaje de activación.

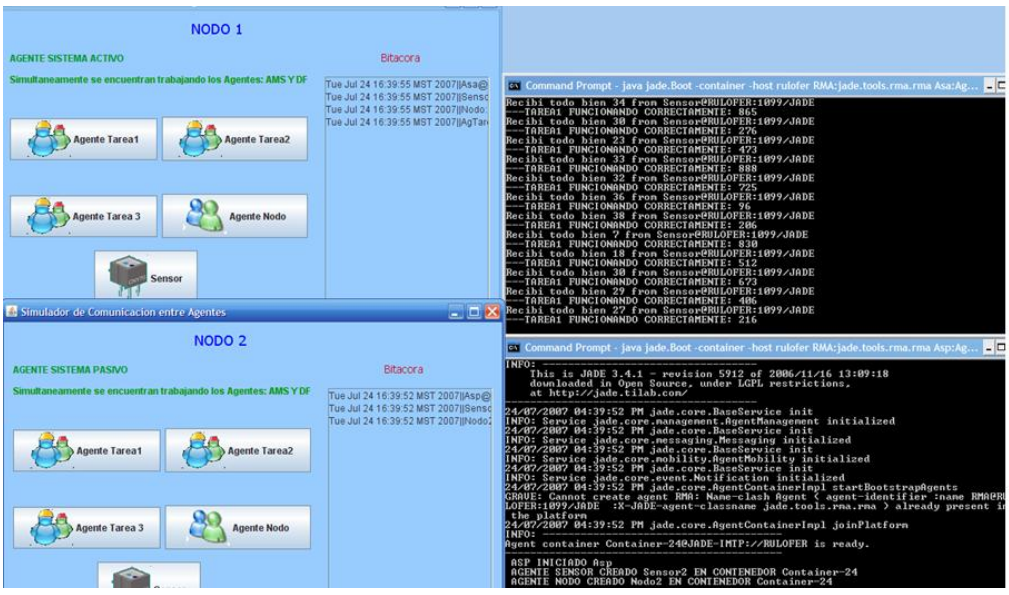


Figura D.5 Sensor funcionando correctamente.

Cuando un dispositivo falla su Agente Nodo, lo aísla y manda mensaje al Agente Sistema de que el dispositivo a su cargo falló. El Agente Sistema busca si existe una réplica en otro nodo. Al encontrarla, el agente sistema manda un mensaje de activación al agente nodo dormido de la réplica; quien a su vez activa el dispositivo replicado. En la figura D.6 se muestra el momento de que el dispositivo falló en el Nodo1, y fue detenido para evitar que se propagara el fallo en el sitema de control, después se activa su réplica en el Nodo2.

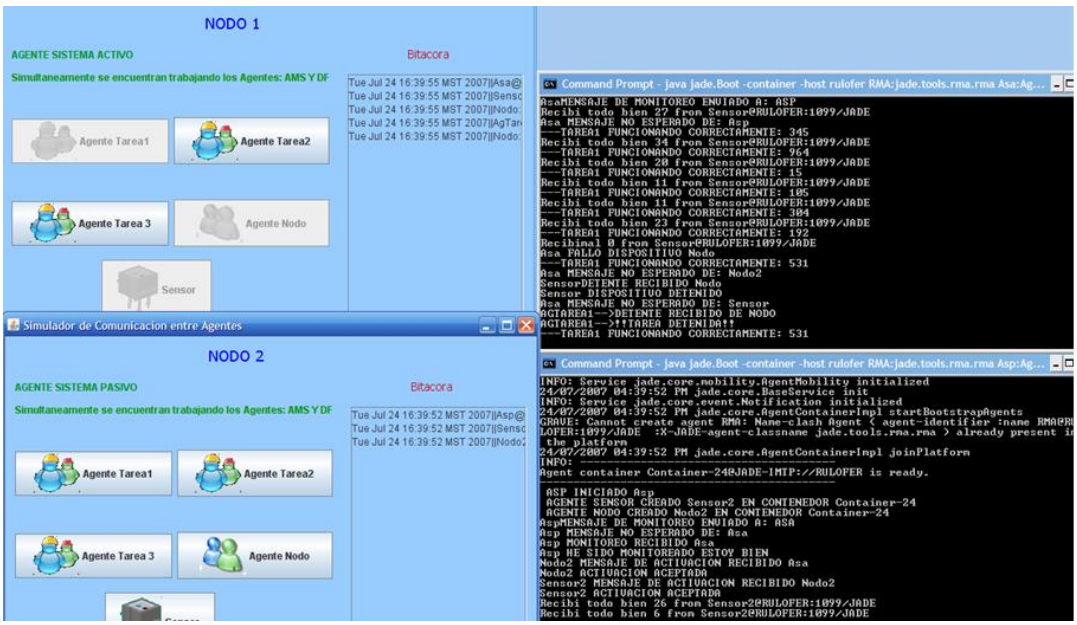


Figura D.6 Fallo en un dispositivo del sistema de control y la activación de su réplica mediante los agentes.

Cuando falla un dispositivo que es una réplica y es crítico el Agente Sistema manda mensaje al usuario, de que se realizará un paro seguro por fallo de dispositivo (x). En la figura D.7 se muestra como el Agente Sistema realiza un paro seguro.

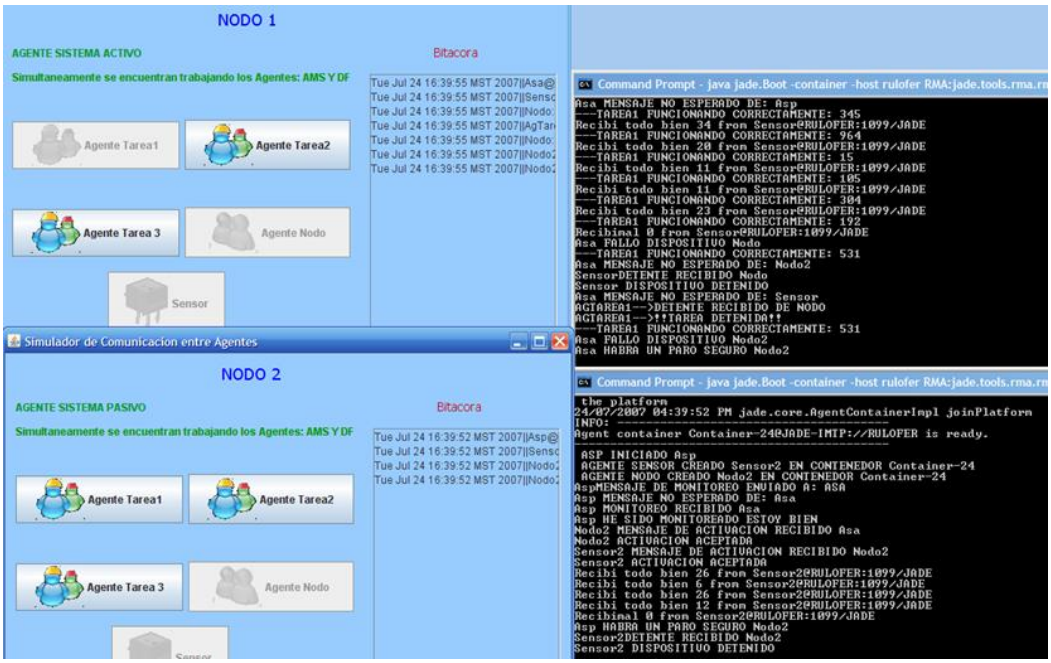


Figura D.7 Fallo de un dispositivo que es réplica.