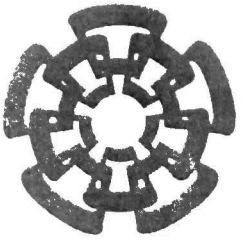
BC-613
Oct.-2011

xx (178959.1)



Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Guadalajara

Protocolos de tolerancia a fallas en sistemas de agentes móviles para la gestión de flujo de trabajo interorganizacional

Tesis que presenta:

Marina Flores Badillo

para obtener el grado de:

Doctor en Ciencias

en la especialidad de:

Ingeniería Eléctrica

Director de Tesis

Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco, Agosto de 2010.

Protocolos de tolerancia a fallas en sistemas de agentes móviles para la gestión de flujo de trabajo interorganizacional

Tesis de Doctorado en Ciencias
Ingeniería Eléctrica

Por:
Marina Flores Badillo
Maestra en Ciencias
CINVESTAV del IPN Unidad Guadalajara 2004-2006

Becario de CONACYT, expediente no. 191083

Director de Tesis
Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, Agosto de 2010.

Protocolos de Tolerancia a Fallas en Sistemas de Agentes Móviles para la Gestión de Flujo de Trabajo Interorganizacional

Resumen

Esta tesis trata sobre tolerancia a fallas en los sistemas automatizados de Flujo de Trabajo Interorganizacional basados en agentes móviles. Los problemas abordados incluyen los aspectos de localización, detección de huérfanos, terminación y recuperación de agentes que están guiando procesos de flujo de trabajo en redes locales e internet.

Para solucionar los problemas mencionados se proponen estrategias de control agentes móviles organizados en dos protocolos: a) un protocolo de localización y terminación de agentes móviles, así como el control de huérfanos y b) un protocolo para la recuperación de agentes móviles. El enfoque está basado en el uso de un agente administrador para cada red perteneciente al Flujo de Trabajo Interorganizacional, el cual se encarga de gestionar los permisos de entrada y tiempo de ejecución de agentes provenientes de otra compañía.

Adicionalmente, se propone una extensión de las redes de Petri (PN- Petri Nets) temporizadas llamada PN con cronómetros globales, la cual es utilizada para especificación y análisis de los protocolos propuestos.

Fault-Tolerant Protocols in Mobile Agent Systems for Interorganizational Workflow Management

Abstract

This thesis deals with fault tolerance in mobile agent based automated Interorganizational Workflow Systems. The addressed problems include issues related to location, orphan detection, termination, and agents guiding workflow processes within local area networks and through internet.

For dealing with the above problems, strategies for the control of mobile agents are proposed. These strategies are organized in two protocols: a) a protocol allowing location of agents, control of agent losses and orphan detection, and b) a mobile agent recovery protocol. The approach is based on using a Management Agent in each net belonging to the Interorganizational Workflow. This agent handles both control access policies and allowed execution time of agents from external companies.

Additionally, this thesis proposes an extension to Time Petri Nets called Global Stopwatch Petri Nets used for the analysis and specification of the proposed protocols.

Agradecimientos

A Dios, por haberme dado el ser, el entendimiento, la salud y por caminar a mi lado en los momentos más difíciles de mi vida.

A mi madre, por su ejemplo, confianza, lucha y constante entrega para que yo pueda lograr mis objetivos, por enseñarme a no darme por vencida cuando las cosas se ponen difíciles y sobre todo por su incondicional apoyo para lograr mis metas.

A mi familia, por confiar en mí y por estar siempre presentes en mi lucha por la superación.

A mi asesor el Dr. Luis Ernesto López Mellado por su gran comprensión y apoyo en los momentos difíciles y tristes, además por orientarme y aportarme sus conocimientos a través del desarrollo de esta tesis.

A todos mis amigos, por brindarme su confianza y ánimo para seguir siempre adelante.

Y finalmente agradezco mucho a CONACYT por el apoyo brindado ya que sin su valiosa ayuda esto no habría sido posible.

INDICE

Introducción 1

Capitulo 1. Automatización Confiable del Flujo de Trabajo 4

1.1 Conceptos Básicos de Flujo de Trabajo 5

1.2 Modelado de Flujo de Trabajo basado en Redes de Petri 6

1.3 Flujo de Trabajo y Tecnología de Agentes 9

1.3.1 Sistemas de Agentes Móviles 9

1.3.2 Beneficios de aplicar la tecnología de agentes a la administración de Flujo de trabajo 11

1.3.3 Clasificación de las aplicaciones de agentes a sistemas de Flujo de Trabajo 12

1.4 Metodología para el desarrollo de WFMS basados en Agentes Móviles 13

1.4.1 Generalidades 13

1.4.2 Metodología de modelado 13

1.4.3 Simulación de los modelos 15

1.4.4 Metodología de automatización de Flujo de trabajo 16

1.4.5 Problemas presentados 17

1.5 Tolerancia a Fallas en los Sistemas de Agentes Móviles 18

1.5.1 Desventajas de los Sistemas basados en Agentes móviles 18

1.5.2 Planteamiento General del problema 19

1.5.3 Mecanismos de Tolerancia a Fallas para Agentes Móviles 20

1.5.3.1 Generalidades 20

1.5.3.2 Protocolos para el Control de Agentes móviles 21

1.5.3.2 Protocolos de tolerancia a Fallas para Agentes móviles 24

1.6 Propuesta de solución 28

Capítulo 2 Protocolos orientados al Control de Agentes 33

2.1 Generalidades 34

2.1.1 Organización o esquema de red de WF-Interorganizacional 34

2.1.2 Elementos que participan en los protocolos 35

2.2 Protocolo de Localización de Agentes 36

2.2.1 Fase 1. Creación de un camino de proxies en una red de WF-Interorganizacional 39

2.2.2 Fase 2. Localización de Agentes 43
2.2.2.1 Localización Local 44
2.2.2.2 Localización Externa 48
2.3 Protocolo de Control de Agentes Huérfanos 52
2.3.1 Gestión del concepto de Energía 53
2.3.1.1 Renovación del tiempo de vida de un Agente móvil local 54
2.3.1.2 Renovación del tiempo de vida de un Agente móvil externo 58
2.4 Protocolo de Terminación de Agentes 62
2.4.1 Eliminación Pasiva de Agentes móviles 62
2.4.2 Eliminación Activa de Agentes móviles 63

Capítulo 3 Protocolo para la recuperación de Agentes Móviles 67
3.1 Tolerancia a fallas de agentes móviles 68
3.2 Protocolo de recuperación de agentes móviles 69
3.2.1 Recolección y almacenamiento de datos para el protocolo de recuperación de agentes 71
3.2.2 Monitoreo de Agentes 73
3.2.2.1 Creación de Agentes testigo 74
3.2.2.2 Fallas en los agentes testigo 76
3.2.2.3 Disminución en el número de Agentes testigo 78
3.2.2.4 Eliminación de Agentes testigo cuando el AgMóvil ha terminado sus tareas dentro de una red. 84
3.2.2.5 Monitoreo de Agentes testigo cuando el Agente móvil viaja a un nodo de una zona distinta 86
3.2.3 Detección de fallas 88
3.2.4. Recuperación de fallas 95

Capítulo 4. Modelado de Protocolos de Control de Agentes 99
4.1 StopWatch Petri Nets 100
4.1.1 Generalidades 100
4.1.2 Definición Formal de las GSWPN 105
4.1.3 Reglas de habilitación y disparo 107
4.1.4 Modelado de un protocolo utilizando GSWPN 110
4.2 Modelado de Protocolos de Control de Agentes 113
4.2.1 Localización de Agentes 114

4.2.2 Terminación de Agentes 119
4.2.3 Control de Agentes Huérfanos 121

Capítulo 5 Análisis de protocolos para el control de agentes 127
5.1. Análisis de modelos en Global StopWatch Petri Nets 128
5.2 Análisis mediante la herramienta PHAVer 128
5.2.1 Descripción de la Herramienta PHAVer 128
5.2.2 Ejemplo de análisis de autómatas utilizando PHAVer 130
5.2.3 Análisis de un caso de estudio utilizando PHAVer 132
5.2.3.1 Autómata con cronómetros (StopWatch Automata-SWA) 132
5.2.3.2 Construcción de un SWA a partir de una GSWPN 133
5.2.3.3 Análisis hacia adelante del caso de estudio 136
5.3 Simulación de los modelos en GSWPN 140
5.3.1 Simulación de un caso de estudio 141
5.3.2 Simulación de los modelos de los protocolos propuestos 145
5.3.2.1 Simulación de los modelos para el protocolo de localización de agentes móviles 145
5.3.2.2 Simulación de los modelos para el proceso de renovación de tiempo de vida de un agente móvil 148
Conclusiones 153
REFERENCIAS 156

Introducción

La automatización de Flujo de Trabajo es un factor muy importante para lograr competitividad entre las empresas con las tecnologías actuales. La automatización involucra actividades tanto humanas como las realizadas por máquinas (van der Aalst & Hee, 2002) para mejorar el desempeño de los procesos y lograr las metas de los negocios con alta eficiencia (Reijers & van der Aalst, 2005). Muchas compañías se han dado cuenta de que los procesos del negocio, dentro de la organización así como entre organizaciones, no han sido claramente descritos y por lo tanto no hay suficientes técnicas y métodos para controlar tales procesos.

El Flujo de Trabajo (**WF** por sus siglas en inglés – Workflow) tiene que ver con la logística de los procesos de una organización y con la tecnología de implementación, para desarrollar sistemas de control que manejen tanto la secuencia de tareas como la asignación de recursos. Hoy en día la tecnología de agentes proporciona soluciones flexibles y distribuidas para la automatización de los procesos del negocio. Algunos trabajos que integran ambos conceptos se encuentran en (Repetto et al., 2003; Savarimuthu et al., 2004; Flores-Badillo & López-Mellado, 2008).

En (Flores-Badillo, 2006) se presenta un método para el modelado y control de la Automatización de WF basado en Agentes Móviles. En esta propuesta, un Agente Móvil guía al proceso a través de las diferentes unidades organizacionales (departamentos), donde varias tareas son ejecutadas de acuerdo a un plan específico para el caso manejado. Este método permite concebir el sistema de WF como un sistema multi-agente; de la misma forma pueden abordarse problemas grandes y complejos de WF Inter-organizacional, donde varios socios de negocios o compañías se integran para compartir procesos de WF (van der Aalst, 1999); en este contexto los agentes tienen que viajar a través de internet con el objetivo de alcanzar un nodo que se encuentra en otra organización y realizar actividades de WF (Figura 1.2).

La motivación principal de utilizar sistemas multi-agentes está en la natural distribución de la información, recursos y acciones. Sin embargo, los agentes móviles introducen nuevos niveles de complejidad ya que operan dentro de un ambiente que es

autónomo, abierto a ataques en la seguridad (dirigidos por agentes o hosts maliciosos) y a fallas para localizar recursos (Pears et al., 2003).

De esta manera, uno de los problemas más importantes a enfrentar con este tipo de enfoques es la posibilidad de fallas en los nodos de la red o en los enlaces y, debido a la importancia de la información que el Agente maneja, la confiabilidad y seguridad de los Sistemas de WF Inter-organizacionales basados en tecnología de Agentes, son muy importantes y presentan aspectos cruciales. Por lo tanto es necesario proporcionar mecanismos que logren estos objetivos.

La confiabilidad de un sistema depende en gran medida de la tolerancia a fallas, que es la capacidad del sistema o componente de continuar con la operación normal a pesar de la presencia de fallas en software o hardware. En un sistema tolerante a fallas las fallas serán enmascaradas o tratadas apropiadamente.

Existen varias técnicas de tolerancia a fallas en sistemas basados en agentes (Baumann, 1999; Pleisch & Schiper, 2000; Marin et al., 2001), pero ninguno se ha aplicado al WF Inter-organizacional como el planteado en la Figura 1.2 y pocos tratan con redes cuyos nodos están separados por la internet.

Esta problemática es abordada en esta tesis y para resolverla se proponen mecanismos de control y recuperación de Agentes Móviles aplicados al WF Interorganizacional. En esta propuesta se definió un formalismo para la especificación y análisis temporal de los mecanismos de control de Agentes la cual llamamos Global StopWatch Petri Nets (**GSWPN**).

La propuesta está basada en la incorporación de un Agente Administrador para cada red perteneciente al WF-interorganizacional, el cual se encargará de manejar a los agentes móviles que se encuentren en ejecución dentro de la red a la que pertenecen. Con este mecanismo se pretende simplificar el manejo y control de los agentes móviles realizando una administración por zonas o porciones de red, donde cada zona representará la red de una compañía o socio perteneciente al WF interorganizacional, lo cual no se realiza en otras propuestas.

Los mecanismos de control y recuperación de agentes móviles se presentan en forma de protocolos; en esta tesis se proponen dos tipos de protocolos: 1) protocolos

orientados al control de agentes móviles (tratando problemas de localización y terminación de agentes móviles, así como control de huérfanos) y, 2) un protocolo para la recuperación de agentes móviles.

Los protocolos propuestos en esta tesis, incluyen extensiones de conceptos utilizados en otras propuestas, los cuales son adaptados al integrar un control en base a varios Agentes Administradores, lo cual nos proporcionará grandes ventajas simplificando la localización de un agente móvil y el control de su tiempo de ejecución dentro del sistema. Además la integración de los protocolos propuestos nos proporciona un mayor control en la detección de la falla para una pronta recuperación del agente móvil.

Además, con la ayuda de las GSWPN se comprueba comportamiento de los protocolos propuestos mediante la simulación de los modelos y el análisis temporizado de los mismos con respecto a los parámetros (temporizaciones) definidos para cada modelo.

Esta tesis se encuentra organizada de la siguiente manera. En el capítulo 1 se presentan conceptos básicos de WF y de sistemas Multi-agentes, así como los problemas y mecanismos de control y tolerancia a fallas de los sistemas basados en agentes. El capítulo 2 presenta la propuesta de manejo y control de agentes en forma de protocolos de control de agentes móviles. En el capítulo 3 se propone un protocolo para la recuperación de Agentes móvil basado en el concepto de *agentes* testigo y *checkpoints*. En el capítulo 4 se presenta un formalismo de modelado, la cual se presenta como una extensión a las Redes de Petri Temporizadas y se modelarán algunos de los protocolos propuestos. En el capítulo 5 se mostrarán técnicas para obtener información de los modelos mediante la simulación de los mismos. Finalmente, se enuncian algunas conclusiones y extensiones al trabajo presentado.

Capitulo 1.

Automatización Confiable del Flujo de Trabajo

Resumen: En este capítulo se presentan algunos conceptos básicos sobre los sistemas de flujo de trabajo. Se hace una revisión de los enfoques más importantes sobre el modelado de flujo de trabajo y de la aplicación de la tecnología de agentes a la coordinación de sistemas de flujo de trabajo. Finalmente se presenta la problemática que motivó la metodología propuesta en esta tesis.

1.1 Conceptos Básicos de Flujo de Trabajo

Los sistemas de información fueron diseñados para soportar la ejecución de tareas individuales. En la actualidad también son utilizados para controlar, monitorear y soportar los aspectos de logística de los procesos del negocio; en otras palabras, necesitan manejar el Flujo de Trabajo (**WF** por sus siglas en inglés – Workflow) a través de la organización.

Desde su uso por primera vez en 1980, el término WF ha ido ganando mucha importancia, especialmente en la mejora de los procesos dentro de las organizaciones (Shi et al., 1998). Muchas organizaciones, con complejos procesos de negocio, han identificado la necesidad de conceptos, técnicas y herramientas para soportar la Administración de los Flujos de Trabajo.

La Administración de Flujo de Trabajo (**WFM** por sus siglas en inglés – Workflow Management), es una tecnología dinámica la cual está siendo explotada por los negocios de una gran variedad de industrias. Su uso más extendido es dentro del ambiente de las oficinas en las operaciones intensivas del personal como seguros, bancos, hospitales, administración legal y general, etc.; se usa también en algunas clases de aplicaciones de la industria y la manufactura (Hollingsworth, 1995).

Hoy en día WFM representa un aspecto fundamental para lograr competitividad entre las organizaciones, donde WF está relacionado con la automatización de los procesos del negocio en el cual documentos, información o tareas son pasados de un participante a otro de acuerdo a un conjunto de reglas predefinido para lograr una meta de negocio (WfMC, 1999; van der Aalst & Hee, 2002; van der Aalst, 1998). El objetivo principal de la WFM es mejorar el desempeño de los procesos y ayudar a logar las metas de negocio con una alta eficiencia (Reijers & van der Aalst, 2005).

De acuerdo a la WorkFlow Management Coalition (**WfMC**), un sistema de WFM es "un sistema que define, crea y maneja la ejecución del WF a través del uso de software ejecutándose en uno o más motores de WF, y el cual es capaz de interpretar la definición del proceso, interactuar con participantes de WF y, cuando sea requerido, invocar el uso de herramientas IT (*Information Technology— Tecnología de Información*) y aplicaciones" (WfMC, 1999).

Esto permite utilizar la tecnología de WF para modelar cualquier tipo de proceso, no solamente en el ámbito de los procesos de negocios, sino en cualquier tipo de procesos que impliquen la colaboración entre personas o entre personas y máquinas.

Los beneficios principales del WF son:

- Eficiencia Mejorada – la automatización de muchos procesos del negocio conlleva a la eliminación de varios pasos innecesarios.
- Mejor control del Proceso – Mejoramiento de la Administración de procesos del negocio logrado a través de la estandarización de métodos de trabajo.
- Flexibilidad – El software de control de procesos permite su rediseño en línea con necesidades del negocio cambiantes.
- Mejoramiento en los procesos del negocio – enfocarse en los procesos del negocio nos lleva a su simplificación.

1.2 Modelado de Flujo de Trabajo basado en Redes de Petri

En general, una metodología de modelado WF no sólo tiene que especificar cómo fluye el trabajo sino que debe abarcar tres perspectivas fundamentales:

- *Perspectiva funcional.* Donde se especifica cómo se constituyen los WF.
- *Perspectiva de comportamiento.* En la que se especifica la forma en que se van a ejecutar las tareas, teniendo en cuenta dos aspectos fundamentales:
    - *Descriptivo.* Donde se indica cómo se ejecutan los procesos, es decir: en serie, paralelo o condicionales.
    - *Prescriptivo.* En la que se especifican las restricciones de ocurrencias de los WF.
- *Perspectiva organizativa.* Donde se deben especificar las políticas de ejecución con respecto a los agentes y las aplicaciones.

En el desarrollo del ciclo de vida de los Sistemas de WF, el marco de modelado es la primera y más importante de las etapas de diseño, pero a pesar de los esfuerzos de la WfMC, carece de una teoría estandarizada que proporcione bases teóricas (van der Aalst, 1998), es por eso que existen muchas propuestas de modelado, cada una con sus ventajas y debilidades en diferentes aspectos (Lu & Sadiq, 2007). Sin embargo, las Redes de Petri (**PN**- Petri Nets) han sido ampliamente adoptadas para el modelado de WF, desde 1977 (Zisman, 1977), debido a su semántica formal, su naturaleza gráfica, la abundancia de técnicas de análisis y verificación de propiedades de modelos de WF estáticos (van der Aalst, 1996; van der Aalst, 1998; van der Aalst & Anyanwu, 1999; Adam et al., 1998; Li et al., 2004), y además han sido extendidas para mejorar expresividad (Eshuis & Dehnert, 2003; van der Aalst & Hofstede, 2005).

A pesar de que las PN proporcionan modelos claros y sin ambigüedades, cuando los sistemas son grandes y complejos, la manipulación y construcción de modelos de PN ordinarios se vuelve una tarea difícil. Es por eso que en (Flores-Badillo, 2006) se propuso una técnica general para el modelado de WF concebido como un modelo multi-nivel (modelado con 3 niveles en el sistema n-LNS), además de estar basado en el paradigma de agentes, donde un agente móvil (AgMóvil) guía al proceso a través de las distintas unidades organizacionales para realizar tareas WF (ver Figura 1.1).

De esta manera, cuando se habla de sistemas de WF donde requieren la integración de varias compañías (socios de negocios) para el procesamiento del mismo caso (WF Interorganizacional) (van der Aalst, 1999), el sistema podría verse como un Sistema basado en Agentes Móviles donde un Agente tiene que viajar de un sistema a otro (de una compañía a otra) a través de internet con el objeto de compartir procesos de negocio (ver Figura 1.2).

Utilizar Agentes para implementaciones de WF trae muchos beneficios (como se verá más adelante) sin embargo, hace necesaria la utilización de otros recursos para dotar al sistema de un control de los agentes y de tolerancia a fallas.

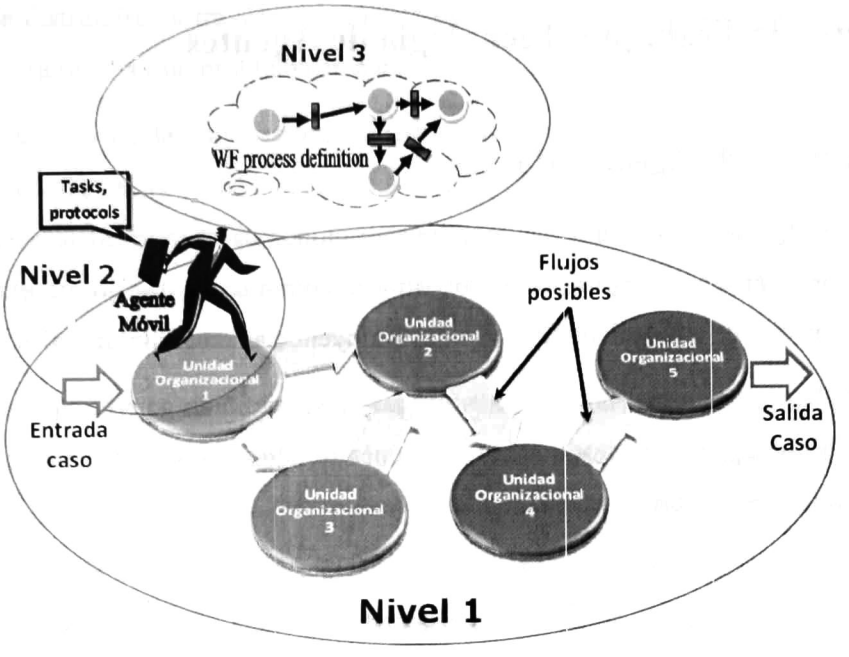


Figura 1.1 Enfoque general de Modelado de flujo de trabajo a 3 niveles

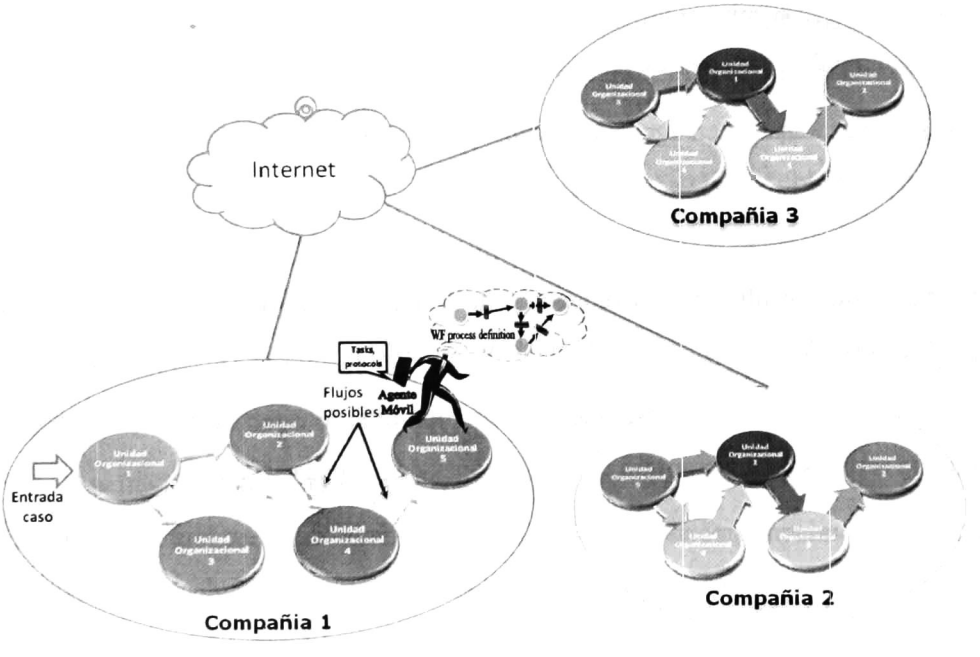


Figura 1.2 Esquema de un Flujo de Trabajo Interorganizacional basado en Agentes Móviles

1.3 Flujo de Trabajo y Tecnología de Agentes

1.3.1 Sistemas de Agentes Móviles

Los agentes Móviles son programas autónomos que pueden viajar de una computadora a otra bajo su propio control; ofrecen un marco de trabajo robusto y eficiente para el desarrollo de aplicaciones distribuidas incluyendo aplicaciones móviles.

De la misma forma, un Agente móvil puede ser definido como un programa que es iniciado en una computadora y que autónomamente migra a varios hosts para realizar las tareas para las que fue programado (Hevia & Vasa, 2000).

Un agente estacionario es ejecutado solo en el sistema donde comenzó su ejecución; si requiere de información de diferentes sistemas o necesita interactuar con otros agentes, éste utiliza comunicación estándar de cliente-servidor (RMI, RPC, CORBA).

Un agente móvil no siempre está ligado al sistema donde inició su ejecución; es capaz de moverse por sí mismo a través de los diferentes nodos de la red donde se aloja, modificando eventualmente su ambiente.

Los sistemas basados en agentes móviles proporcionan muchas ventajas debido a la movilidad, por ejemplo:

a) Interacción directa con el recurso mediante la migración al sitio donde éste se encuentra, lo que resulta en una reducción de la latencia de la red y en el ancho de banda utilizado (Cabri et al., 1998; Hevia & Vasa, 2000).

b) Interacción con los usuarios mediante la migración al sitio donde éstos se encuentran, obteniendo respuestas más rápidas a las peticiones de los usuarios.

c) Se ejecutan de manera autónoma y asíncrona; los agentes móviles, una vez creados, se vuelven independientes del proceso que los creó y pueden operar de manera asíncrona y autónoma.

d) Se adaptan dinámicamente; los agentes móviles pueden percibir su ambiente de ejecución y reaccionar autónomamente a los cambios en el mismo. Tienen la habilidad

única de distribuirse a través de los hosts de una red para mantener la configuración óptima para resolver un problema en particular.

La mayoría de las aplicaciones distribuidas caen naturalmente en el modelo de sistemas de agentes móviles debido a que los agentes pueden migrar de manera secuencial a través de la red, pueden enviar en forma paralela a otros agentes a visitar diferentes computadoras y pueden permanecer estacionarios e interactuar con recursos remotos, etc. Algunas de las aplicaciones en las que se utilizan agentes móviles son:

- Recuperación distribuida de información (distributed information retrieval), en lugar de mover grandes cantidades de datos a los motores de búsqueda, las aplicaciones pueden crear agentes y enviarlos a las fuentes remotas de información donde los agente localmente crean índices de búsqueda que pueden posteriormente ser transportados al sistema de origen.
- Servicios de Red de Telecomunicaciones. Configuraciones de red dinámicas utilizan agentes móviles para mantener el sistema flexible y efectivo.
- Comercio Electrónico (E-commerce). Una transacción comercial puede requerir acceso en tiempo real a recursos remotos, como cotizaciones de bolsa o tal vez negociaciones agente-agente.
- Asistencia personal. La habilidad de los agentes móviles para ejecutarse en hosts remotos los hace adecuados como asistentes realizando tareas en la red a nombre de sus creadores.
- Diseminación de información. Los agentes pueden diseminar información, como noticias o actualizaciones automáticas de software para vendedores.

La flexibilidad de la computación basada en Agentes móviles no se presenta sin desventajas, actualmente los principales obstáculos del desarrollo de agentes es la confiabilidad y la seguridad de los mismos (Hevia & Vasa, 2000).

1.3.2 Beneficios de aplicar la tecnología de agentes a la administración de Flujo de trabajo

La tecnología para la WFM promete proporcionar una forma eficiente para modelar y controlar procesos de negocio complejos dentro y entre las organizaciones. Los beneficios de esta tecnología incluyen:

1. Definición de procesos explícita
2. Rápida reacción en ambientes cambiantes
3. Fácil seguimiento de operaciones

Como la administración de WF se enfoca en manejar la lógica de los procesos, es necesario integrar otras tecnologías para controlar totalmente los procesos del negocio incluyendo la asignación de actividades y la distribución de recursos. La tecnología de agentes proporciona soluciones flexibles, distribuidas e inteligentes para la Administración de los procesos del negocio.

En (Yan et al., 2001) se toma la definición de agente como "un sistema computacional situado en algún ambiente y que es capaz de acción autónoma en ese ambiente con el objetivo de lograr sus objetivos de diseño" Diferentes definiciones pueden encontrarse en (Wooldrige, 2002; Nwana, 1996). Además, enumeran los beneficios de aplicar la tecnología de agentes a la administración de los proceso del negocio:

1. Arquitectura Distribuida. Proporciona una estructura de para integrar sistemas administradores distribuidos de procesos del negocio.
2. Automatización. La autonomía inherente de los agentes software pueden satisfacer actividades como substitución de una persona. Los agentes pueden iniciar WF basados en eventos.
3. Interacción. Los agentes software permiten organizaciones para interactuar unos con otros por medio de una semántica de intercambio de mensajes.
4. Administración de Recursos. Los agentes pueden representar recursos.
5. Reactividad. Los agentes pueden reaccionar a circunstancias cambiantes y tienen la habilidad de generar caminos alternativos de ejecución.

6. Interoperación entre sistemas heterogéneos. Los agentes pueden ser heterogéneos, la interacción recae en una semántica de mensajes para intercambiar definiciones planes y servicios.
7. Hacer decisiones inteligentes, algunas características de alto nivel de los agentes, como la capacidad de aprendizaje, son muy útiles en la administración de WF.

Un Agente Inteligente es capaz de operar autónomamente y de poseer un comportamiento flexible con el objetivo de lograr sus metas de diseño; además, tiene las propiedades de reactividad, pro-actividad y habilidad social (Wooldridge, 2001).

1.3.3 Clasificación de las aplicaciones de agentes a sistemas de Flujo de Trabajo

En (Yan et al., 2001) se clasifican las aplicaciones de agentes a sistemas de WFM, en dos formas:

1. Administración de WF Agente-Aumentado. En este escenario los agentes son como servicios proporcionados por el sistema administrador de WF, el objetivo de usar agentes es incrementar la automatización de los sistemas WF. Desde el punto de vista del sistema, los agentes no necesariamente interactúan unos con otros, el motor de WF controlar sus acciones.
2. Administración de WF Basado en Agentes. En este escenario la lógica del proceso esta embebida en los agentes, los agentes son independientes entre ellos y responsables de la ejecución de los procesos; el proceso del negocio en su totalidad está formado por subredes en estos agentes.

Ambas tecnologías se integran en (Repetto et al., 2003; Marin & Brena, 2005; Wang et al., 2005; Savarimuthu & Purvis, 2004; Savarimuthu et al., 2004), pero en ninguna de ellas abordan el problema de control de agentes o tolerancia a fallas.

1.4 Metodología para el desarrollo de WFMS basados en Agentes Móviles

1.4.1 Generalidades

En trabajos previos (Flores-Badillo, 2006; Flores-Badillo et al., 2009), se presentó una metodología para el desarrollo de Sistemas de WF basados en el paradigma de Agentes. La idea principal se basó en concebir el WF como un Sistema donde un Agente Móvil guía el proceso de WF a través de las diferentes unidades organizacionales, en las cuales varias tareas son ejecutadas de acuerdo al caso a procesar.

Durante la fase de diseño los componentes fueron descritos de forma clara y compacta utilizando un formalismo basado en Redes de Petri Multinivel (n-LNS) (Flores-Badillo & Lopez-Mellado, 2008). El uso de n-LNS permite descripciones modulares y jerárquicas del Sistema de WFM. Los modelos se definieron de manera separada y después fueron relacionados a través del etiquetado de las transiciones, con lo cual, se permitía la actualización de los modelos de manera sencilla debido a la modularidad y a la flexibilidad del modelo completo. n-LNS permite modelar sistemas largos exhibiendo comportamientos más complejos que los que se obtendrían utilizando PN ordinarias.

1.4.2 Metodología de modelado

El sistema se describe como un conjunto de unidades organizacionales interconectadas que tienen una distribución específica de los recursos disponibles. El comportamiento del agente es especificado de acuerdo a:

a) La descripción del comportamiento general del agente y conocimiento común para todos los agentes: nombre del agente, operaciones básicas y protocolos de interacción (colaboración, competición por recursos, etc.).

b) Descripciones particulares del comportamiento específico del agente, como el plan de tareas y mapa de accesibilidad el cual describe el proceso asignado y el acceso permitido a las respectivas unidades organizacionales.

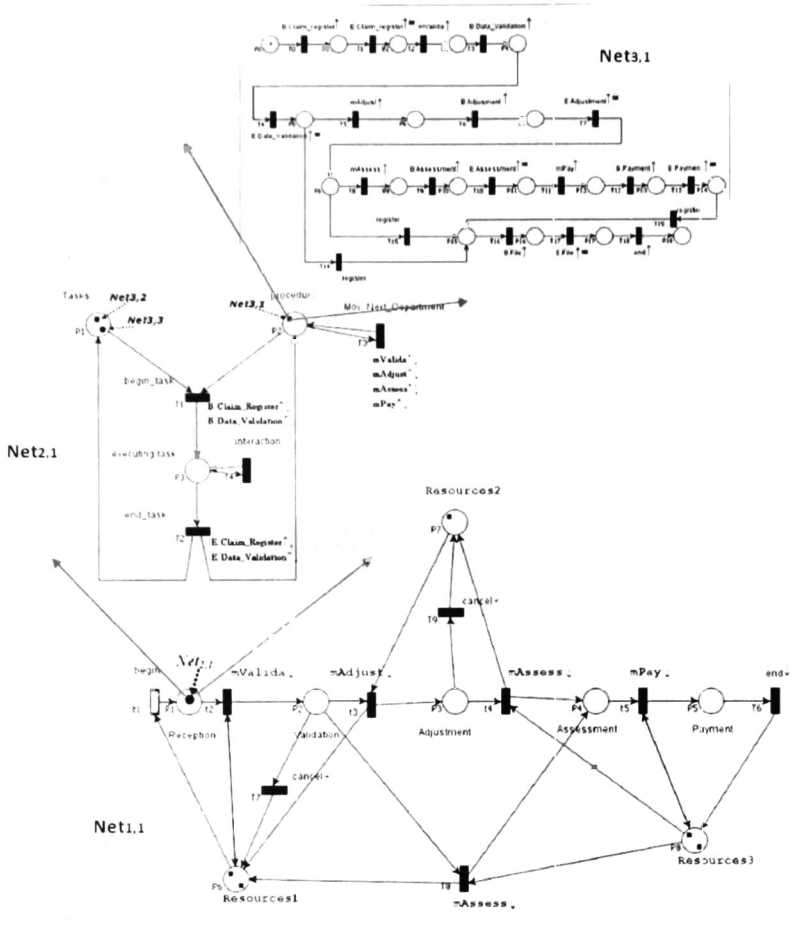


Figura 1.3 Modelo a 3 niveles para el WF del procesamiento de una queja utilizando n-LNS

El Sistema de WF se modela utilizando tres niveles en el formalismo n-LNS (Sánchez-Herrera, 2004), modelando en cada nivel una parte distinta del sistema. El método fue ilustrado a través de un caso de estudio que trata con el proceso de una queja en una compañía de seguros. En la Figura 1.3 se muestra el modelo a tres niveles del WF del caso de estudio, donde la $Net_{1,1}$ (nivel más alto) describe la estructura general de la compañía y todos los posibles flujos de información entre las unidades organizacionales; la $Net_{2,1}$ (segundo nivel) modela el comportamiento general de un Agente Móvil que guía al proceso de WF a través de la compañía de acuerdo a una *definición del proceso de WF* la cual es modelada con la PN $Net_{3,1}$.

1.4.3 Simulación de los modelos

A pesar de que la construcción de los modelos es fácilmente realizada siguiendo la metodología, cuando la complejidad y la cantidad de las actividades del sistema crecen, el tamaño de los modelos se incrementa. Debido a la ausencia de procedimientos analíticos para la verificación del correcto funcionamiento de los modelos resultantes, la simulación interactiva resulto ser una adecuada solución para validar los modelos obtenidos (Flores-Badillo & López-Mellado, 2010). Esta tarea se llevó a cabo mediante el uso de una herramienta de simulación llamada MASGAS (Padilla, 2008), la cual permite la edición visual y la ejecución interactiva de modelos multinivel expresados bajo el formalismo n-LNS (Sánchez-Herrera, 2004).

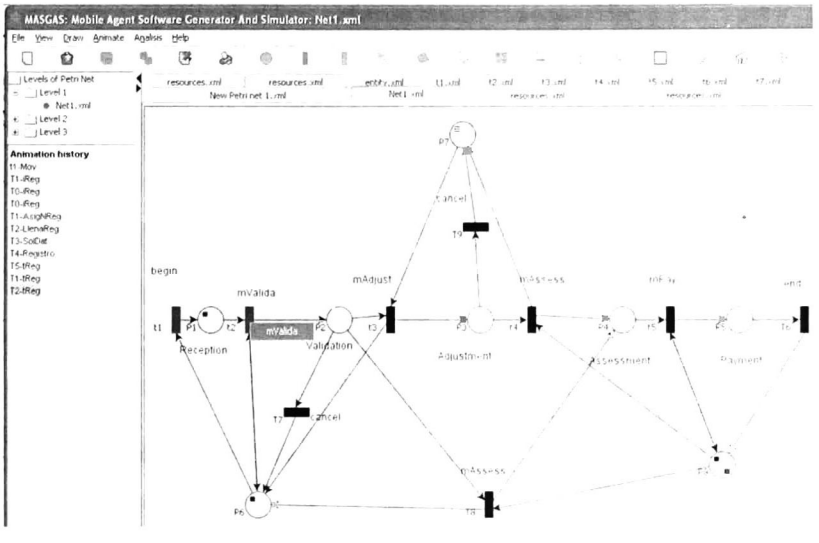


Figura 1.4 Ventana de simulación de la herramienta MASGAS.

Basado en n-LNS, MASGAS proporciona un control sintáctico completo sobre los diferentes niveles jerárquicos del sistema de red, con lo cual se evita el anidamiento de redes de niveles superiores, permitido por otros simuladores (Kummer et al., 2002). En la Figura 1.4 se muestra la red que describe la estructura de la compañía (la red $Net_{1,1}$ en la Figura 1.3) donde las tareas de WF serán realizadas, el token en *P1* representa la red que modela el comportamiento general de la entidad que guía el caso de WF En la figura se muestra que la transición *t2* está habilitada respecto a la etiqueta *mValida*.

1.4.4 Metodología de automatización de Flujo de trabajo

Los componentes del modelo basado en PNs multinivel sirven como guía para definir una implementación en software del WF utilizando Java y el middleware JADE (Bellifemine et al., 2007). Esta fase es soportada por una guía para el desarrollo de software, permitiendo la definición de componentes Java para la especificación de sistemas de agentes en la fase de diseño. El software obtenido es distribuido en un conjunto de computadores en red que manejaban la migración de los agentes móviles.

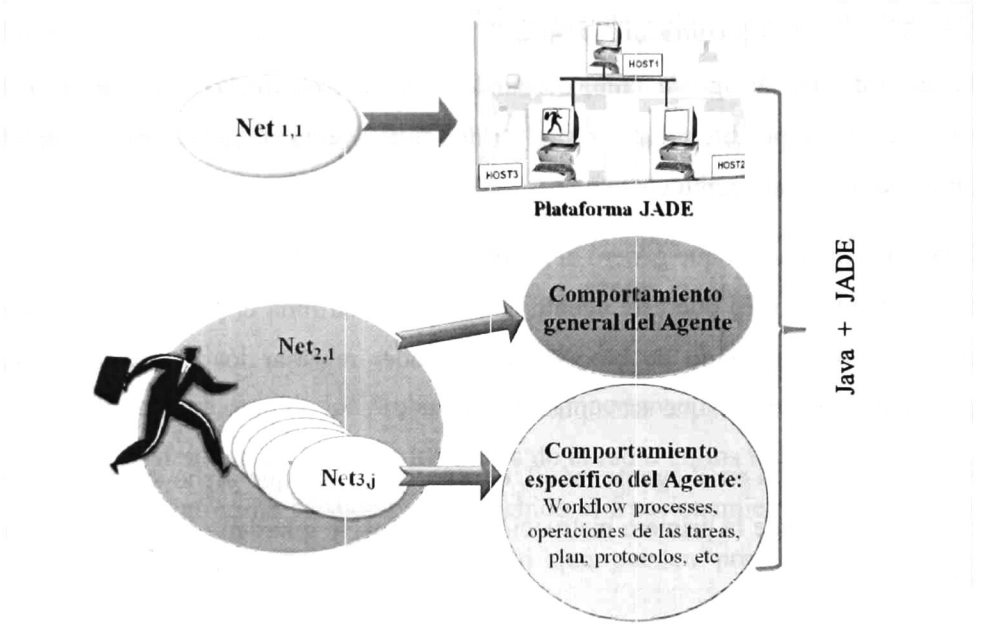


Figura 1.5 Esquema general para la automatización de WF

En la Figura 1.5 puede observarse que el ambiente del agente es definido mediante la red de nivel 1 que representa la estructura general de la compañía, donde cada lugar de la red puede ser representado por un *Contenedor* (host) en JADE (Bellifemine et al., 2007). El comportamiento general del agente que guía el proceso de WF es definido mediante la red de nivel 2, el cual es creado como una subclase de la clase genérica *Agent* de JADE; el comportamiento particular de dicho agente es descrito como un *behaviour* en JADE y fue determinado por la definición del proceso de WF ($Net_{3,1}$ en Figura 1.3) el cual, debido a que sigue el comportamiento de una Máquina de estados finitos (**FSM** por sus siglas en inglés-

Finite State Machine), se utilizó el *FSMBehaviour()* de JADE para definir el comportamiento que seguiría.

1.4.5 Problemas presentados

La implementación obtenida utilizando las guías presentadas, tiene varias desventajas relacionadas con la confiabilidad de los procesos, entre las que se encuentran:

- El sistema fue implementado en una red de computadoras local, y no soporta la detección del agente móvil cuando éste falla.
- En caso de que el agente falle, o cuando la aplicación que crea al agente móvil lo requiera, no es posible localizar el sitio donde el agente se encuentra ejecutando o el último sitio que el agente visitó.
- Cuando la aplicación que generó al agente falla, no se soporta la terminación del agente móvil creado, por lo que este agente, una vez que termina el proceso para el cual fue creado, entra en estado de bloqueo al no poder regresar los resultados y continúa utilizando recursos sin que éstos puedan liberarse.
- Si ocurre una falla en el agente móvil, el proceso de WF que dicho agente guiaba tiene que iniciarse desde la primera instrucción ya que no se guardan estados intermedios del agente móvil.
- No se soporta seguridad en los agentes ni en los recursos, por lo que no hay políticas de utilización de dichos recursos.

La metodología propuesta para el desarrollo de Sistemas de WF es una primera etapa para la automatización de procesos de negocio complejos en grandes industrias. Sin embargo, en compañías donde las unidades organizacionales o departamentos se encuentran distribuidas en varias ciudades, el agente móvil tiene que viajar a través del medio que conecta dichas redes (como internet) enfrentándose a nuevos problemas (como se describirá en la siguiente sección). Por lo tanto, capacidades más sofisticadas deben ser incluidas en los agentes y en su ambiente (como protocolos de seguridad, control de pérdida de agentes, etc), para soportar los problemas que se presentan en este tipo de configuraciones.

1.5 Tolerancia a Fallas en los Sistemas de Agentes Móviles

En esta sección se presenta de manera general el problema a abordar y se hace una revisión de trabajos relacionados a este problema

1.5.1 Desventajas de los Sistemas basados en Agentes móviles

La motivación principal de utilizar sistemas multi-agentes está en la natural distribución de la información, recursos y acciones. Sin embargo, los agentes móviles introducen nuevos niveles de complejidad ya que operan dentro de un ambiente que es autónomo, abierto a ataques en la seguridad (dirigidos por agentes o hosts maliciosos) y a fallas para localizar recursos (Pears et al., 2003). Además, debido a que los agentes móviles son programas distribuidos, son susceptibles a varios tipos de anomalías: fallas en los procesos, fallas de comunicación y fallas debido a errores en la programación (Hevia & Vasa, 2000).

De esta manera, uno de los problemas más importantes a enfrentar con este tipo de enfoques es la posibilidad de fallas en los nodos de la red o en los enlaces. Las fallas en el sistema pueden llevar a una pérdida parcial (debido al fraccionamiento temporal de la red por ejemplo) o completa del agente, de aquí que pueden presentarse las siguientes situaciones:

a) La aplicación que creó al agente "cree" que el agente se ha perdido (ha fallado y terminado su ejecución) cuando en realidad no es así. Si la aplicación creara a otro agente para que realice las mismas operaciones, puede causar que existan dos agentes realizando múltiples ejecuciones de la misma tarea en diferentes sitios.

b) La aplicación que crea al agente espera a que éste termine su ejecución, pero el agente ha fallado, lo que claramente nos lleva a una situación de bloqueo.

c) La aplicación falla, por lo que el agente creado por la misma puede estar consumiendo recursos que ya no son necesarios debido a que su aplicación propietaria (*owner*) falló.

De esta forma pueden existir varias situaciones de falla que provocarían un funcionamiento incorrecto del sistema. Por lo tanto, son necesarios mecanismos de control de agentes que traten con estos tipos de anomalías y proporcionar la capacidad de tolerancia a fallas.

1.3.2 Planteamiento General del problema

El marco de trabajo se sitúa en el contexto de la automatización de WF Interorganizacional, donde un grupo de compañías colaboran para procesar el mismo caso de WF. El sistema es concebido bajo el enfoque de Agentes Móviles, donde un agente móvil guía a un proceso de WF a través de las diferentes unidades organizacionales, por lo tanto, se tiene a un agente móvil para cada caso viajando entre los diferentes departamentos (entre los nodos o sitios de la red) de una compañía, guiando un caso de WF hasta que éste es procesado (ver Figura 1.1).

En un sistema de WF Interorganizacional (van der Aalst, 1999), un agente móvil guiará un caso de WF a través de los departamentos (sitios/nodos/unidades organizacionales) que pueden pertenecer a un socio de negocio diferente y, debido al esquema de modelado, al nodo de una red distinta.

Se supone que cada socio de negocio tiene una distribución de red propia y que la comunicación entre las diferentes redes se realiza a través de una red abierta como internet (ver Figura 1.2), donde no se tiene control en los nodos intermedios.

Se pretende que este Sistema basado en Agentes móviles soporte cierto grado de tolerancia a fallas en donde:

1. Sea posible localizar la red y el sitio donde el agente se encuentre en ejecución en un momento determinado.
2. Se soporte la detección y eliminación de agentes huérfanos que ya no son requeridos y por lo tanto, la liberación de los recursos que se encuentren utilizando innecesariamente.
3. En la presencia de la falla durante la ejecución del agente móvil, que éste pueda ser iniciado en el estado más cercano al que tenía antes de ocurrir la falla.

4. Que se soporte la notificación de los casos en los que no se pueda recuperar el estado del agente.
5. Que el sistema notifique las fallas que se presenten.

Existen varios autores que trabajan con este tipo de problemas, cada uno basado en un protocolo o enfoque diferente (como se describirá en la siguiente sección). Sin embargo, la mayoría de los trabajos revisados manejan la administración de las fallas en una sola red sin hacer distinción entre la migración de un agente dentro de una red local o la migración del agente a través de internet. En esta tesis se proponen protocolos para proporcionar la capacidad de tolerancia a fallas en los sistemas que tengan una distribución utilizada en las redes de WF Interorganizacional. Las estrategias incorporadas en estos protocolos consideran el sitio donde los agentes se ejecutan, es decir, dentro de una red local o en ambientes abiertos (internet).

Las estrategias utilizan variantes de algunos de los conceptos utilizados en los protocolos de tolerancia a fallas presentados en la siguiente sección.

1.5.3 Mecanismos de Tolerancia a Fallas para Agentes Móviles

1.5.3.1 Generalidades

Debido a que fallas en los sistemas de Agentes Móviles pueden ocasionar una pérdida parcial o completa de los agentes, la tolerancia a Fallas se ha convertido en un aspecto fundamental en estos sistemas y para el desarrollo de aplicaciones que se basen en dichos sistemas (Pleisch & Schiper, 2001).

La tolerancia a Fallas puede definirse como la habilidad de los sistemas o componentes de continuar con una operación normal a pesar de la presencia de fallas en software o hardware. También puede ser definida como el número de fallas que un sistema o componente puede manejar antes de que la ejecución normal sea debilitada o dañada (Pears et al., 2003).

Existe una gran atención dentro de la comunidad que investiga métodos de tolerancia a fallas en los agentes móviles, concerniente a la perdida de agentes móviles en

servidores remotos que fallan (Pears et al., 2003), pero primero se debe asegurar una correcta detección de la falla (Pleisch & Schiper, 2001).

En esta sección se describen algunos de los protocolos existentes para proporcionar tolerancia a fallas para agentes móviles. Esta revisión se divide en dos partes, en protocolos para el control de agentes (orientados a la localización de agentes, terminación de agentes y detección de agentes huérfanos) y en protocolos orientados a la recuperación de agentes.

1.5.3.2 Protocolos para el Control de Agentes móviles

Las principales tareas que realiza un sistema de control de agentes son las siguientes:

a) *Localización de Agentes*. Permite determinar el sitio donde el agente se está ejecutando.

b) *Terminación de Agentes.* Permite terminar la ejecución de los agentes cuando ésta ya no es necesaria (terminación activa o pasiva).

c) *Detección de huérfanos.* Permite detectar cuando el *owner* (quien inició al agente) del agente no está disponible para evitar este agente haga uso innecesario de recursos (ver Figura 1.6).

La terminación de agentes y la detección de huérfanos en sistemas de agentes son funciones muy importantes, ya que los agentes que se encuentran en ejecución utilizan recursos que son valiosos tanto para el usuario como para el sistema. El sistema tiene una cantidad limitada de recursos así que es importante minimizar el costo de ejecución de los agentes, sobre todo si su computación ya no es requerida. La detección de huérfanos garantiza que, aún si falla el mecanismo de terminación del agente, el ahora innecesario agente puede ser detectado por el sistema y terminar su ejecución. Por lo tanto, la detección de huérfanos significa determinar si un agente es aún necesitado por la aplicación que lo creó, y si no es así, el agente es un huérfano y debe de suspenderse su ejecución.

En (Baumann, 1999) Baumann propuso un algoritmo basado en los conceptos *de sombras, conceptos de energía* y *camino de proxies;* para evitar el problema de la existencia de agentes huérfanos que consuman recursos de manera innecesaria.

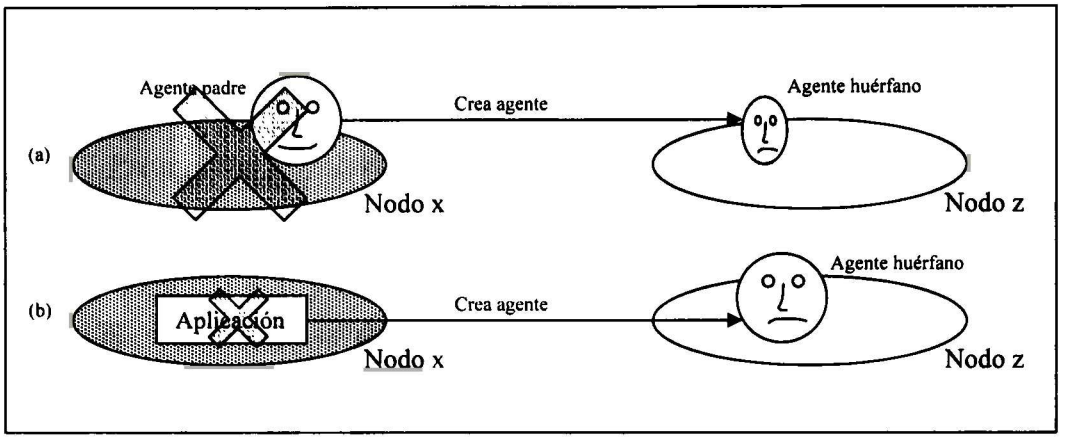


Figura 1.6 Agentes huérfanos (a) cuando el nodo donde se encontraba el agente padre colapsa, (b) cuando falla la aplicación que crea al agente.

Existen varios protocolos propuestos que proporcionan las funcionalidades antes mencionadas y se clasifican precisamente de acuerdo a los conceptos utilizados por Baumann.

1) El concepto de Energía soporta la *detección de huérfanos* para agentes móviles. El protocolo garantiza que un agente huérfano solo vive por un periodo corto de tiempo. Cada agente tiene una limitada cantidad de energía para consumir, cada acción que realiza o cada recurso que utiliza consume su energía. El protocolo fue introducido por primera vez por Baumann (Baumann, 1999) y después implementado por primera vez por Jochum (Jochum, 1997).

2) El concepto de camino de proxies. Soporta la *localización y terminación de agentes*. Este protocolo es bien conocido en el área de los sistemas distribuidos, algunos autores que han usado este concepto son: Fowler en (Fowler, 1985), para la localización de objetos en sistemas distribuidos, Andrew Black en (Black et al., 1986) y Erick Jul en (Jul et al., 1988) para la localización de agentes móviles, y Emeral y Awerbunch (Awerbuch & Peleg, 1995) para localización de usuarios móviles. De una forma similar, Baumann usó este protocolo para localizar agentes móviles en un sistema multi-agente (Baumann, 1999). En este concepto, cada agente deja un rastro o

camino en el sistema, cada que el agente migra deja información en el nodo actual sobre el nodo siguiente a donde migrará.

3) El concepto de sombras soporta la *localización y terminación de agentes móviles y también la detección de huérfanos*. Baumann en (Baumann, 1997) combina el concepto de caminos y de energía para este protocolo e introduce algunas variaciones para el protocolo de sombras. La combinación resultante deja al agente con la mayor parte de su autonomía y proporciona tolerancia a fallas. El concepto de sombras utiliza la idea de tener un *marcador de posición* o *sombra* el cual es asociado por el sistema de agentes a cada nuevo agente. La sombra mantiene el registro de la localización de todos los agentes dependientes. En (Padilla, 2007) se propone un algoritmo basado también en estos tres conceptos, pero no maneja una estrategia adecuada para la renovación de tiempos de vida ni migración del agente Controlador (sombra), ya que no hace una distinción entre el tipo de nodo al que se viajará, es decir, si se trata de un nodo dentro de la red local o si tiene que migrar en un ambiente abierto como internet donde, los retrasos que pudieran recibir los mensajes son mayores.

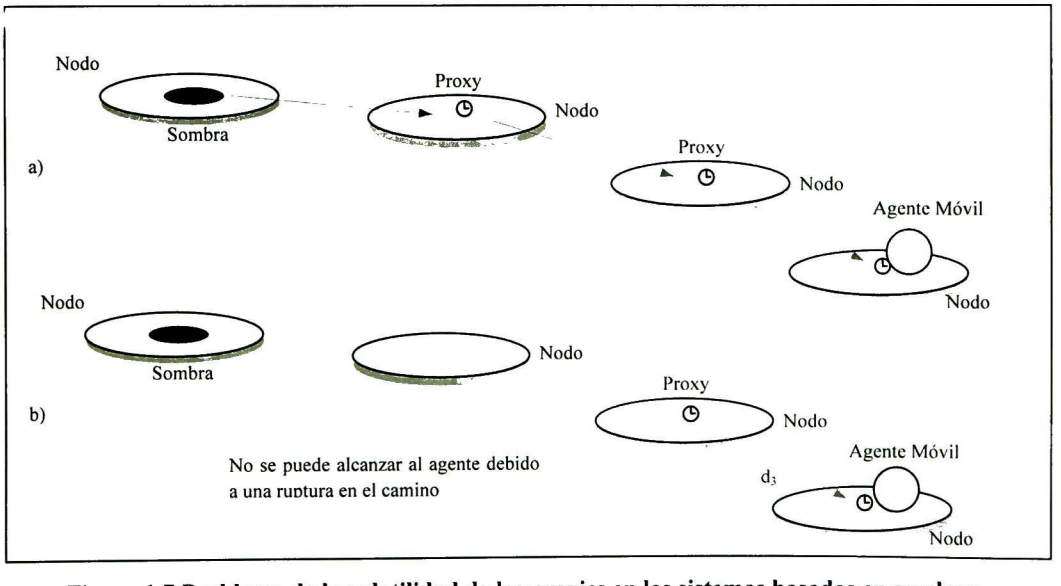


Figura 1.7 Problema de la volatilidad de los proxies en los sistemas basados en sombras.

Aunque el protocolo de sombras de (Baumann, 1999) es una buena solución para el control de agentes móviles, cuando se utiliza en aplicaciones implementadas bajo internet,

es posible que todo el sistema colapse debido a la volatilidad del camino de proxies; lo que ocasiona que el camino sea inaccesible en un cierto punto (ver Figura 1.7).

1.5.3.2 Protocolos de tolerancia a Fallas para Agentes móviles

La tecnología de Agentes móviles ha sido utilizada para muchas aplicaciones sensibles a faltas o errores, entre ellas se encuentra el comercio electrónico, administración de sistemas, etc (Rothermel, & Strasser, 1998). Ya que los agentes poseen una inherente autonomía, no existe una entidad que naturalmente realice el monitoreo del progreso de la ejecución de los agentes, por lo tanto el agente puede perderse o bloquearse debido a fallas en los nodos donde el agente se encuentra ejecutando, o puede perderse durante alguna migración a un nuevo nodo. Afortunadamente existen algunas propuestas que proporcionan confiabilidad en los sistemas de agentes móviles, algunas de las propuestas existentes se mencionan a continuación.

- *Protocolos basados en Replicación de agentes*

La replicación de datos y/o computación, es una buena forma de obtener tolerancia a fallas en sistemas distribuidos. Un componente de software replicado es definido como un componente que posee una representación en dos o más hosts (Marin et al., 2001). Hay dos tipos principales de protocolos de replicación:

1) Replicación Activa, en la que todas las replicas procesan todo mensaje de entrada de manera concurrente.

2) Replicación pasiva, en el cual los mensajes de entrada son procesados por una sola réplica la cual transmite su estado actual de manera periódica al resto de las réplicas para mantener consistencia.

Las estrategias basadas en replicación activa nos llevan a que la misma tarea se ejecute varias veces consumiendo recursos de manera innecesaria y violando la propiedad de *exactamente-una ejecución* (exactly-once execution) (Spector, 1982) que requieren algunos sistemas; de hecho, muchas de las aplicaciones basadas en agentes móviles, requieren que el agente se ejecute exactamente una vez. Por ejemplo, tomemos el caso en el

que un agente móvil tiene asignada la tarea de realizar una reservación en un hotel y en un vuelo para un viaje de negocios, se espera que el agente haga ambas reservaciones de ser posible y mandar mensajes de su estado al usuario. El agente debe garantizar que se hagan las reservaciones exactamente una sola vez ya que hacer varias reservaciones para el mismo usuario sería un error, por lo que la replicación activa no es una opción para este tipo de aplicaciones y mecanismos que proporcionen esta característica son muy importantes. Sin embargo, la replicación activa tiene la ventaja de que proporciona una recuperación rápida del sistema.

La replicación pasiva economiza la utilización del procesador activando las replicas redundantes únicamente cuando ocurre una falla. Por lo tanto requiere menos recursos que la replicación activa, pero requiere la implementación de mecanismos de votaciones o acuerdos para que las réplicas redundantes determinen que la réplica activa ha fallado; de esta forma se podrá decidir cuál será la réplica que se ejecutará y cuales permanecerán inactivas.

En (Marin et al., 2001), se presenta una estrategia de replicación basada en la idea de cambiar, durante tiempo de ejecución del sistema, la estrategia que se utilizará para replicar, es decir, elegir de manera dinámica qué replicar y cuántas veces replicarlo, así como el tipo de replicación a utilizar, ya sea replicación activa o replicación pasiva. De la misma forma, en (Ahn & Min, 2005) se utiliza la replicación de agentes para soportar la tolerancia a fallas, pero lo que se replican son los agentes que proporcionan servicios, no los agentes móviles que utilizan dichos servicios; no garantiza la tolerancia a fallas de los agentes móviles, pero sí garantiza que el sistema no se bloquee cuando un servicio falle.

En (Silva & Popescu-Zeletin, 1998; Pleisch & Schiper, 2001; Pleisch & Shiper, 2000), se utiliza la estrategia de replicación pasiva para soportar la tolerancia a fallas y dicen respetar la propiedad de *exactamente-una ejecución*. Desafortunadamente los trabajos basados en replicación pasiva requieren una actualización constante del estado de las réplicas, por lo que el intercambio de mensajes se incrementa, además necesitan una estrategia, muchas veces complicada, para que al final de cada fase se decida cuál es el estado del agente que va a replicar y cuál de las réplicas será la réplica líder o réplica activa. Más aún, en presencia de fallas, necesitan una estrategia de votación y consenso que

determine de manera correcta que ha ocurrido una falla y cuál será la nueva réplica activa. De la misma forma, es necesaria una entidad encargada del proceso de votación, y asumen que esa entidad se encuentra en un sitio confiable (por lo que no soportan la falla de este sitio), así como una correcta validación de la identidad de las réplicas. Debido a que los procesos de votación basan su decisión en el voto de la mayoría, si fallan también cierto número de réplicas pasivas, no se garantiza la recuperación de la falla. Más aún, si hay una segmentación de la red, los algoritmos no siempre detectan que se trata de una segmentación y no de una falla de un sitio, por lo que puede presentarse el caso de tener múltiples copias del agente ejecutando el mismo plan violando la propiedad de *exactamente-una ejecución*.

En (Rothermel & Strasser, 1998), también se propone la replicación pasiva como estrategia de tolerancia fallas, pero a diferencia de los trabajos previos mencionados, incluye el uso de un Administrador de transacciones para garantizar que no se ejecuten múltiples copias del mismo agente móvil. El protocolo utiliza transacciones de colas de mensajes a la entrada de cada nodo, lo que permite una comunicación asíncrona entre los procesos en el mismo nodo o en nodos diferentes. Esta estrategia dice asegurar que los mensajes (pudiendo ser también agentes) se entregan una sola vez, independientemente de los errores en los nodos o fallas en la comunicación. La desventaja es que debe suponerse la confiabilidad del sitio donde se almacenan los mensajes o los agentes antes de ser entregados a su destinatario, de la misma forma se consumen muchos recursos para almacenar a las réplicas y requiere la integración de un protocolo de votación donde se asuma la confiabilidad del orquestador y por tanto, puede presentar los mismos problemas que los trabajos antes mencionados.

En este tipo de estrategias, si la aplicación que inició al agente móvil requiriera localizarlo en algún momento, esto sólo sería posible si el agente móvil continuamente enviara su ubicación a la aplicación, por lo que el intercambio de mensajes se vería incrementado. Además, la replicación tiene la desventaja de que, si la aplicación que generó al agente móvil (el cual se replica ya sea pasiva o activamente) tiene una falla crítica, la operación del agente móvil creado ya no es necesaria y sin embargo se estarían consumiendo un gran número de recursos de manera innecesaria, no solo por parte del agente móvil sino por cada una de sus réplicas.

- *Protocolos basados en Testigos*

Una técnica popular para tratar de mantener la disponibilidad de un agente móvil consiste en dejar una réplica (sombra o testigo) de dicho agente móvil en el sitio actual antes de que éste migre a otro sitio. En (Pears et al., 2003) y (Xu & Pears, 2006), los autores utilizan el enfoque de sombras para soportar tolerancia a fallas, además de utilizar el manejo de excepciones. En esta propuesta tanto el servidor como el agente móvil enumeran su conjunto de excepciones y le asocian un manejador, el cual es una función para manejar la excepción presentada. El inconveniente de esta propuesta es que está orientada a aplicaciones basadas en recolección de información (*information retrieval*) por lo que el caso en que tengamos múltiples copias del mismo agente realizando la misma tarea no es un problema. Solo hay una sombra o testigo que verifica la disponibilidad del agente móvil actual, así que cuando tanto agente móvil como su testigo fallan, el agente debe iniciar todo su itinerario.

Otro trabajo que utiliza este concepto es (Lyu & Wong, 2003), donde no solo se utiliza un solo testigo del agente móvil, sino una cadena de testigos y maneja el caso de fallas en alguno de estos testigos. Sin embargo fallas de múltiples testigos no es soportada, además no soporta la recuperación de un sitio que falló y luego vuelve a estar disponible. Para lograr lo anterior algunos autores utilizan un concepto llamado *logs*, en el cual, se guarda un registro de cada una de las operaciones realizadas por el agente móvil en un nodo y dicha información puede ser utilizada para recuperar a un agente que falló hasta la última operación registrada en un sitio. En (Strasser & Rothermel, 2000), presentan un mecanismo para deshacer las operaciones que un agente ha realizado (*rollback*) hasta alcanzar un punto de control llamado *checkpoint* utilizando los logs registrados; sin embargo, no todas las operaciones que realiza un agente pueden "deshacerse" es decir, no se puede aplicar una operación compensatoria para dejar a los recursos del nodo en el estado que estaba antes de que el agente móvil realizara su ejecución. Por lo tanto, no siempre es posible realizar un rollback, además, no soporta el caso de terminar agentes cuando éstos ya no son necesarios.

Elegir entre un protocolo de tolerancia a fallas y otro, depende en gran medida del caso o tipo de aplicación para el cual se elija.

1.6 Propuesta de solución

Algunas de las características importantes de las propuestas mencionadas en la sección anterior, para el control de agentes móviles y tolerancia a fallas son:

- El concepto de energía para el control de agentes, la cual es una buena estrategia para la detección de agentes huérfanos ya que garantiza que un agente huérfano vivirá solo un periodo corto de tiempo, con lo que se previene la existencia de Agentes maliciosos que se ejecuten infinitamente.
- El concepto de sombras permite separar, de la aplicación, las tareas de control de la ejecución de los agentes móviles al utilizar un objeto dependiente, el cual tan pronto es eliminado por la aplicación, todos los agentes dependientes son eliminados eventualmente por el sistema. Sin embargo, tiene la desventaja de que, cuando un AgMóvil desea renovar tu tiempo de vida y dicho objeto se encuentra a una distancia tal que el mensaje de renovación no alcanza a llegar al agente, el AgMóvil termina con su ejecución. Por lo que un ajuste en los tiempos de espera que haga distinción entre una renovación a través de una red local o a través de una red abierta propensa a sufrir altos retrasos, es requerido.
- En cuanto a los mecanismos de tolerancia a fallas se tiene que :
    - Las propuestas basadas en replicación son una buena forma de obtener tolerancia a fallas, sin embargo consumen muchos recursos sobre todo cuando se trata de replicación activa, además no son adecuadas para las aplicaciones que requieran que se considere la propiedad *exactamente-una ejecución* de los agentes móviles. Además, cuando se trata de replicación pasiva, requiere de la implementación de mecanismos para la detección de la réplica activa, votación, validación e identificación de réplicas, así como una actualización constante del estado de las réplicas.

      Las estrategias basadas en testigos tienen la ventaja de brindar un monitoreo constante sobre la ejecución del agente móvil en un sitio anterior, sin embargo, si tanto AgMóvil como su testigo sufren de una falla, ésta no puede ser detectada.

- El utilizar logs para respaldar las operaciones que realiza un AgMóvil es una buena estrategia para así poder recuperar el estado del agente hasta antes de ocurrir una falla. La desventaja es que el estado que se alcanza no siempre es el mismo que tenía el AgMóvil, además si el estado del agente no puede ser recuperado, requiere la implementación de mecanismos para deshacer las operaciones que había realizado el AgMóvil durante su ejecución en el sistema, lo cual no siempre puede lograrse.

Para el problema abordado en esta tesis, la tolerancia a fallas se define en términos de los siguientes requerimientos:

1. Que la aplicación pueda localizar al agente móvil creado, en el momento deseado.
2. Que si la aplicación falla o la operación del agente se ha vuelto innecesaria, que se soporte la terminación de la ejecución del agente.
3. Que se pueda monitorear el estado del agente para asegurar que sigue en ejecución y cuando ésta falle, que pueda recuperarse el estado del agente hasta donde sea posible.
4. Que se soporte la tolerancia a fallas en redes asociadas a un flujo de trabajo interorganizacional (redes de distintas compañías trabajando juntas para procesar un mismo caso de WF y comunicadas a través de internet).

Ahora, para abordar la solución al problema planteado en esta tesis, partiremos de las siguientes suposiciones:

H1. Se respetará lo más posible la propiedad *exactamente-una ejecución* de los agentes, por lo que se supondrá que múltiples ejecuciones de un mismo agente nos llevarían a un estado de error.

H2. Se trabajará con sistemas distribuidos donde los procesos se comunican a través del intercambio de mensajes.

H3. Los procesos (o agentes) solamente terminan su ejecución debido a la presencia de alguna falla repentina, por lo que se supone que no existen procesos maliciosos o hosts maliciosos.

H4. Se supone que todos los hosts visitados por el agente móvil poseen la plataforma necesaria para permitir la ejecución de dicho agente.

H5. Se supone que cada organización (participante de negocio del WF-Interorganizacional) tiene su propia distribución de red, pero toda red tendrá un único lugar para comunicarse con otras redes (ver Figura 1.8), este lugar especial (al que llamaremos *nodos gateway*) será la única entrada/salida para dicha red. Por lo tanto, si un agente quiere entrar/salir de una red, tendrá que pasar primero por este lugar.

H6. Se supondrá la confiabilidad de los sitios determinados como *Gateway* de cada red.

La propuesta de dotar la capacidad de tolerancia a fallas a los Agentes móviles consiste en dos tipos de protocolos:

1. **Protocolos orientados al control de agentes,** donde se abordarán los problemas de: *localización de agentes, control de agentes huérfanos y terminación de agente, y,*
2. **Protocolos orientados a la recuperación de agentes,** se realiza la *recuperación del estado del agente móvil* cuando ocurre una falla en la ejecución del mismo.

Los protocolos se presentarán como el resultado de la interacción de Agentes con diferentes roles. Los diferentes elementos que estarán interactuando se ilustran en la Figura 1.9.

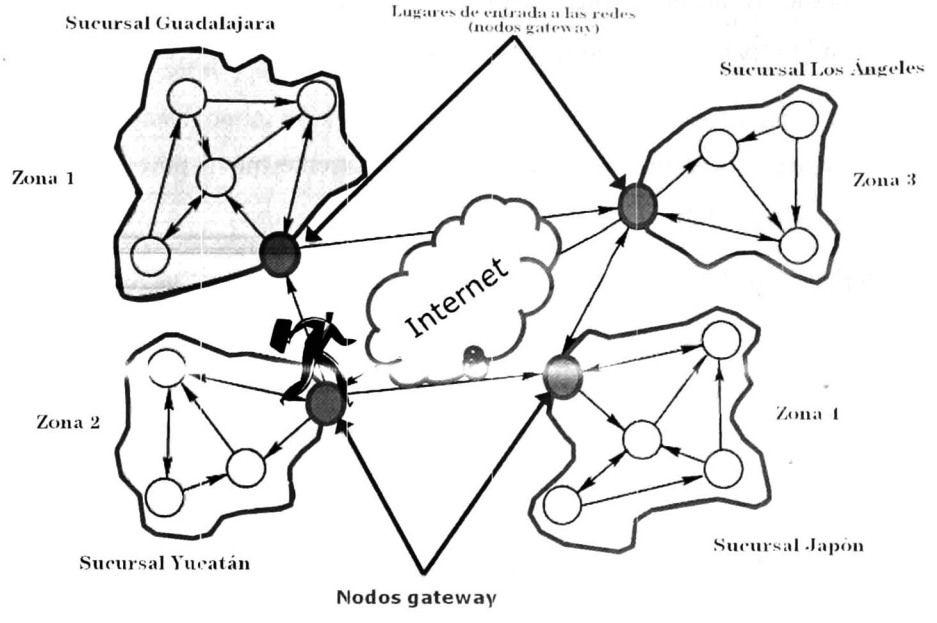


Figura 1.8 Organización de las Redes de los participantes de negocio del WF-Interorganizacional.

Una vez definidos los protocolos para tolerancia a fallas en Agentes Móviles para aplicaciones de WF-Interorganizacional, se modelarán dichos protocolos utilizando una **extensión de las Redes de Petri Temporizadas** (*Stop Watch Petri Nets*) definida en el capítulo 4 para efectuar un análisis temporal de los protocolos propuestos verificando su comportamiento (ver Figura 1.10).

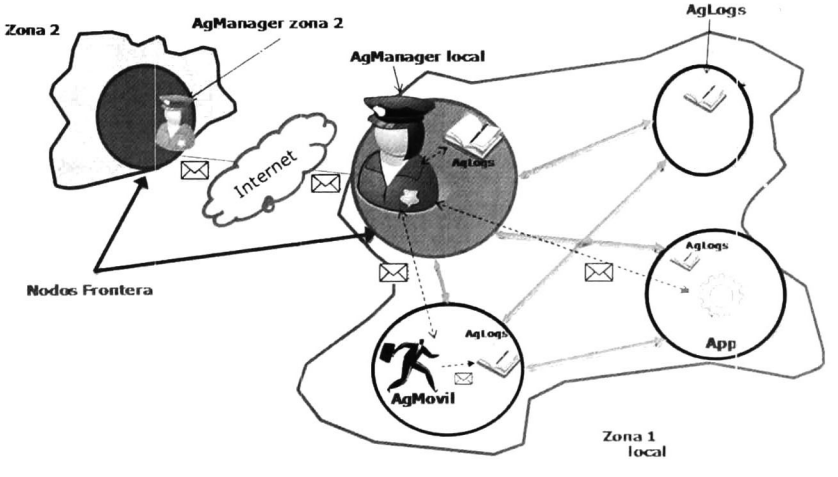


Figura 1.9 Elementos principales para los protocolos de tolerancia a fallas propuestos

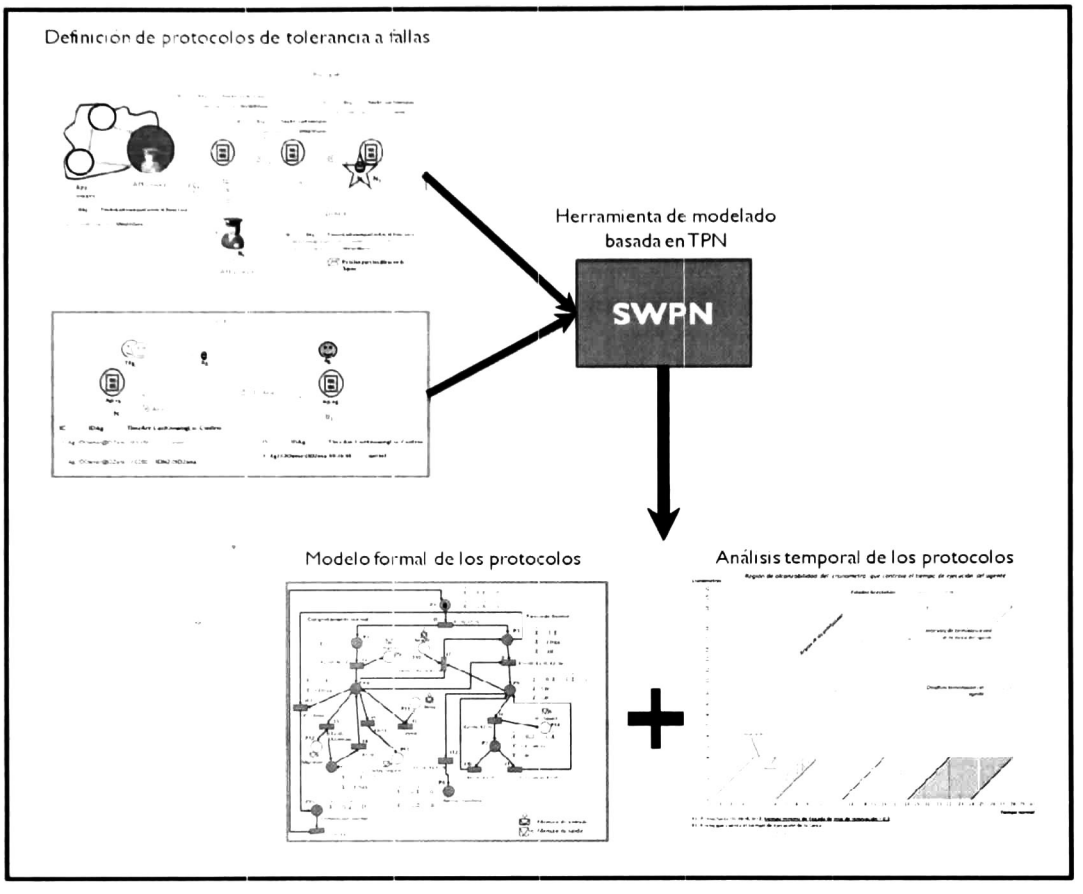


Figura 1.10 Organización general de la propuesta de solución

Capítulo 2

Protocolos orientados al Control de Agentes

Resumen. En este capítulo se describen los diferentes elementos que interactúan en los protocolos propuestos así como la organización de las redes de WF-Interorganizacional. Se proponen protocolos para el control de agentes, cuyas funciones son: la localización de agentes, la detección de agentes huérfanos y la terminación de agentes.

2.1 Generalidades

El control de Agentes en esta tesis, se refiere a: 1) determinar o conocer el sitio donde un agente móvil se encuentra en ejecución en un momento dado; 2) determinar cuándo la aplicación propietaria de un agente (la aplicación que inició el Agente móvil) ya no es accesible y por lo tanto la ejecución del agente se hace innecesaria (detección de agentes huérfanos); y 3) terminar con la ejecución de un agente, ya sea porque se detectó que se trata de un agente huérfano o porque la aplicación así lo necesita.

Como se mencionó en el capítulo 1, los protocolos propuestos para el control de agentes (y para la recuperación del estado del agente, tratado en el siguiente capítulo), se basan en la interacción de componentes con un rol o propósito específico, los cuales se encuentran distribuidos en los nodos o sitios de las redes que componen el Sistema de WF-Interorganizacional.

2.1.1 Organización o esquema de red de WF-Interorganizacional

Como se mostró en la Figura 1.2, en esta tesis se trabaja con problemas que involucran a varios socios de negocio para procesar un mismo caso de WF (WF-Interorganizacional). Por lo tanto, para esta tesis suponemos que:

1. Cada organización posee su propia distribución de red, en la cual, cada sitio o nodo de la red puede representar a un departamento distinto de la organización (Flores-Badillo, 2009).
2. Las distintas redes de las organizaciones se comunican entre sí por medio del intercambio de información a través de una red abierta (como Internet).
3. Cada red involucrada en el WF-Interorganizacional, posee un único lugar de comunicación (con otras redes) o de entrada/salida, el cual se conocerá como lugar *Gateway* (ver Figura 1.14).

2.1.2 Elementos que participan en los protocolos

Como ya se mencionó, los protocolos coordinan la interacción de diferentes elementos distribuidos sobre estas redes. Los elementos que interactúan en los protocolos propuestos se presentan en la Figura 1.15, los cuales son:

1.  *Aplicación.* Es la entidad que controla los casos de WF y la cual se encarga de iniciar el procesamiento de cada *caso* asignándolo a un Agente Móvil. Dicho Agente tendrá como meta guiar el caso de WF, a través de los nodos (o sitios) de los diferentes socios de negocio que conforman el WF-Interorganizacional. A esta entidad es a quien el Agente Móvil le regresa los resultados obtenidos para los cuales fue creado. Cada vez que una aplicación inicia o crea a un Agente Móvil, tiene que asociarlo con un manejador (o administrador de su ejecución). A este manejador se le conoce con el nombre de Agente Administrador (AM). A la aplicación que crea a un agente para guiar un WF se le conoce como *aplicación propietaria (AppOwner)* de dicho agente móvil.

2.  *Agente Móvil (AgMóvil).* Es la entidad creada por una aplicación para guiar el procesamiento de un caso de WF-Interorganizacional. Es asociado a un AM local cuando inicia su ejecución por primera vez dentro de una red. Su nombre dentro del sistema tiene el siguiente formato:

   IDAg:IDOwner@IDZona

   donde: *IDAg* es el Identificador o nombre dado al AgMóvil; *IDOwner* es el identificador del sitio donde se encuentra su aplicación propietaria; y *IDZona* es el identificador de la zona donde fue creado el Agente.

3.  *Agente Administrador (AM).* Es la entidad que se encarga de administrar a todos los agentes que se encuentran en ejecución dentro de su red local. Existe un único AM en la red de cada organización y se ejecuta en el nodo de entrada de la red. Administra la ejecución los agentes que fueron creados en su red por una aplicación local, y de los AgMóviles creados en otras redes pero que se encuentran en ejecución dentro de su red local. Posee una tabla especial (*tabla de agentes*) donde registra la información

necesaria para administrar a los AgMóviles. Se comunica con otros AM de las redes del WF-Interorganizacional para compartir recursos y administrar agentes.

4. *AMLog.* Es un tipo especial de Agente Administrador asociado a cada nodo de la red. Su función principal es la de administrar y registrar las entradas y salidas de AgMóvil al sitio donde se encuentran. De igual forma, registra cada operación, tarea, intercambio de mensaje, etc., realizada por un agente en dicho nodo.

5. *Nodo de la Red.* Es un sitio de una red perteneciente a un socio de negocio en un WF-Interorganizacional, el cual posee los recursos y plataforma necesarios para ejecutar a los diferentes AgMóvil que guían un caso de WF. Cada red posee un tipo especial de nodo que se llama *nodo Gateway*, el cual es el nodo de entrada/salida de cada red, por lo tanto, cuando un AgMóvil desee entrar/salir a la red de alguna organización tendrá que ser a través de éste nodo. Cuando un AM necesite información de la red de una organización tendrá que hacerlo a través de la interacción con el AM local a dicha red. Cada sitio posee un ID único (*IDSitio*) dentro de la red a la que pertenece. Si se necesita referir a un sitio de una zona en específico tendrá que hacerse por medio de: *IDSitio@IDZona.*

La interacción de estos elementos se realiza por medio del paso de mensajes entre ellos, ya sea dentro de la red de cada organización del WF-Interorganizacional o entre redes.

En este capítulo se describirán las operaciones que realiza cada entidad en la definición de cada protocolo de control de agentes.

2.2 Protocolo de Localización de Agentes

Localizar un Agente se refiere a conocer la información del sitio o nodo donde el AgMóvil deseado se encuentra en ejecución en un momento determinado. En una aplicación de WF-Interorganizacional, es necesario además conocer a qué red (compañía, socio de negocio) pertenece el sitio donde encuentra el agente.

Este protocolo se compone de dos fases principales:

Fase 1. Creación de un camino de proxies

Fase2. Utilización del camino creado en la fase 1 para localizar a un agente móvil.

Como ya se mencionó en el capítulo 1, un enfoque comúnmente utilizado en la localización de agentes es el concepto de proxies. El enfoque consiste en dejar, en cada nodo de la red visitado, información sobre el sitio siguiente que visitará un agente durante el proceso de migración (Ver Figura 2.1). De tal manera que si se requiere conocer la ubicación de un AgMóvil, simplemente hay que seguir el rastro o camino formado en cada nodo.

A pesar de ser un enfoque ampliamente utilizado, tiene la desventaja de que el camino de proxies es susceptible a fallas. Entre mayor sea el número de nodos visitados por el agente, mayor es la posibilidad de que el camino se fraccione debido a la ocurrencia de una falla en alguno de los nodos que forman dicho camino (ver Figura 2.2).

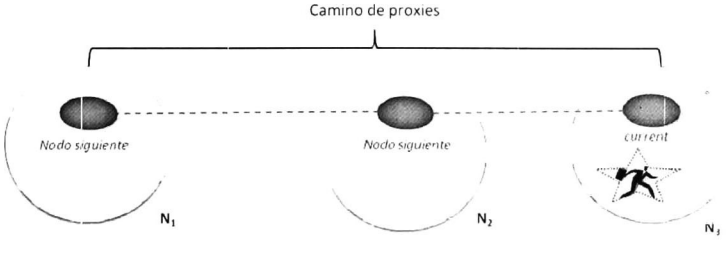


Figura 2.1 Camino de proxies que va formando un AgMóvil durante el proceso de migración

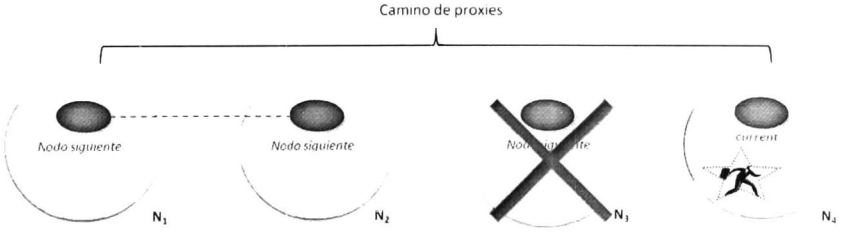


Figura 2.2 Camino de proxies fracturado debido a una falla en el Nodo 3

Sin embargo, para disminuir la posibilidad de hacer el camino inaccesible, usualmente se realizan actualizaciones de la ubicación del agente en periodos determinados de tiempo. De esta forma, el camino para localizar a un agente es acortado.

Esta estrategia es adoptada en esta tesis para localizar AgMóvils en redes implementando WF-Interorganizacional. En este protocolo intervienen las siguientes entidades:

1) Aplicación propietaria: al crear a un AgMóvil, lo asocia al AM local quien se encargará de localizarlo en el momento que así se requiera.

2) AM: cuando un AgMóvil es creado, tiene que registrarse con este agente, el cual registrará su nombre (*IDAg:IDOwner@IDZona*), información del último sitio conocido del AgMóvil (*LastKnowingLoc*), etc. Para ello hace uso de la tabla ilustrada en la Figura 2.3.

3) AMLog:, este agente posee tablas para registrar el arribo del agente al nodo que administra, así como información sobre el sitio (*LastKnowingLoc*) a donde migrará el AgMóvil en su siguiente paso (ver Figura 2.4).

ID	IDAg	TimeArr	LastKnowingLoc	Confirm	ttl	Status	Local
1	Ag1:IDOwner@IDZona	00:01:00	current		50	I	
2	Ag2:IDOwner@IDZona	0:02:00	IDSitio@IDZona	Y	10	inTransit	N

Figura 2.3 Ejemplo de tabla de agentes que utilizan los AM para manejar Agentes Móviles

ID	IDAg	TimeArr	LastKnowingLoc
1	Ag1:IDOwner@IDZonax	0:01:00	*N2@IDZonax*
2	Ag2:IDOwner@IDZonax	0:02:00	*current*

Figura 2.4 Ejemplo de tabla que utilizan los AMLog para manejar el arribo y salida de los Agentes móviles

En la Figura 2.5 se muestran los pasos que se siguen cuando una Aplicación crea a un AM.

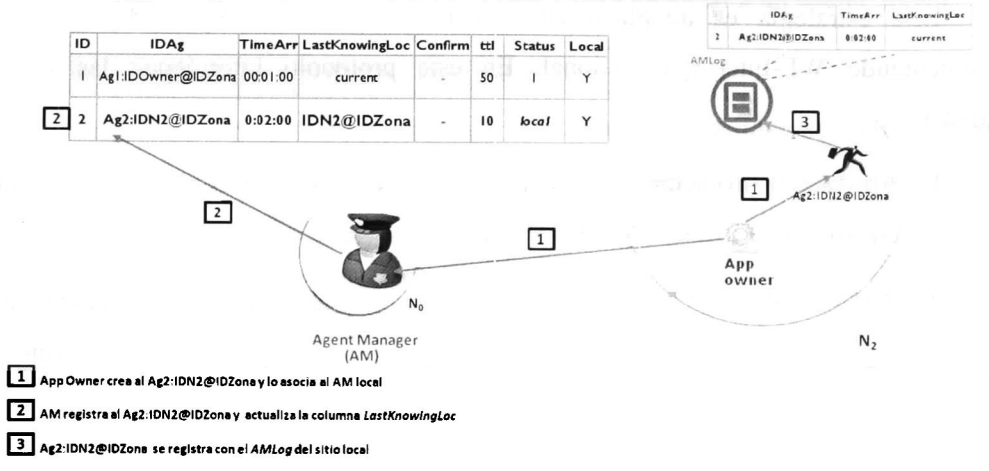


Figura 2.5 Creación de un AgMóvil y asociación del Agente con el AM de la red local

2.2.1 Fase 1. Creación de un camino de proxies en una red de WF-Interorganizacional

Antes de localizar a un AgMóvil, primero tiene que prepararse el camino de proxies a seguir para su localización. El camino se forma ejecutando el siguiente algoritmo:

Algoritmo para la creación de un camino de proxies

1) La *AppOwner* crea a un AgMóvil y lo asocia al AM local (paso 1 Figura 2.5).
2) El AM registra en su *tabla de agentes* (paso 2 Figura 2.5) el ID del agente, el ID del sitio donde el AgMóvil es iniciado e información para reconocerlo como Agente Local (que fue creado dentro de su red).
3) El AgMóvil registra su entrada al sitio donde se encuentra, mediante intercambio de mensajes con el *AMLog*, indicando su tiempo de arribo (paso 3 Figura 2.5).
4) Cuando el AgMóvil termina su ejecución en ese sitio, determina cual es el sitio siguiente a donde migrará, entonces:
   a. Si el sitio siguiente pertenece a la red local,
      4.1) el *AMLog* actualiza la columna de *LastKnowingLoc* de su tabla de registro, colocando en ese registro el identificador del sitio (*IDSitio@IDZona*) a

donde migrará el AgMóvil (paso 4.1 Figura 2.6).

4.2) El AgMóvil migra al sitio siguiente (paso 4.2 Figura 2.6).

b. Si no

4.1) El *AMLog*, actualiza la columna *LastKnowingLoc* de su tabla de registro, colocando el identificador del nodo *Gateway* de la zona local, debido a que si el AgMóvil desea abandonar la red, tendrá que pasar por el nodo de salida de la red (nodo Gateway).

4.2) El AgMóvil migra al sitio donde está el AM.

4.3) El AM local registra la entrada del AgMóvil al nodo y en seguida actualiza el valor de *LastKnowingLoc* en la tabla de agentes indicando el identificador del nodo Gateway de la red a donde viajará el agente y el identificador de la zona (o red) a la que pertenece ese nodo (*IDGateway@IDZona*). Además en la columna *Status*, cambia el valor a *inTransit* para indicar que el AgMóvil se encuentra en tránsito en otra red.

4.4) El AgMóvil registra su salida con el AM y migra a la red del sitio siguiente (ver Figura 2.7).

5) Continúa con el paso 3.

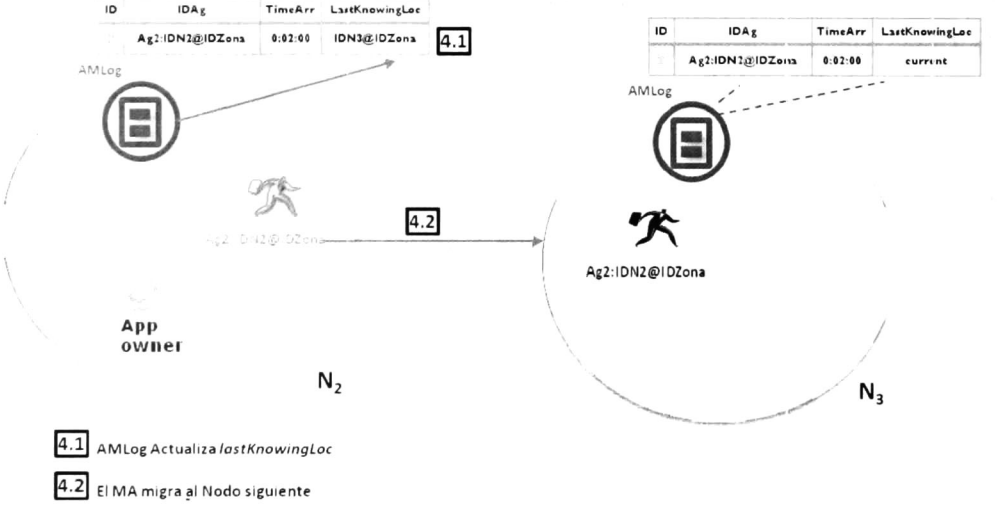


Figura 2.6 Migración de un AgMóvil a un sitio local

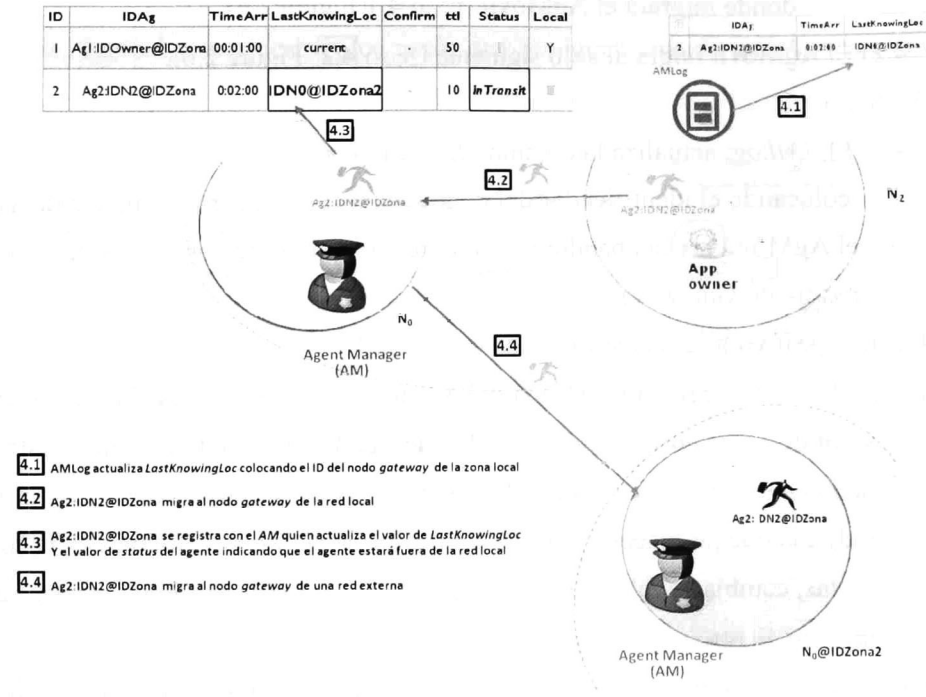


Figura 2.7 Migración de un AgMóvil a un sitio de una zona externa

En la formación del camino de proxies, se considera que cuando un AgMóvil visita de nuevo a un sitio que ya se había visitado antes, la información también es agregada a la tabla. Como cada registro posee además el tiempo en que el AgMóvil llegó al nodo, cuando se desee localizar a un agente se revisará la información más actual que se posea del AgMóvil, es decir, se revisará la que tenga un tiempo de arribo mayor. De esta forma pueden obtenerse: información de todos los sitios que el agente ha visitado (todo el camino de proxies considerando los ciclos formados al visitar el mismo nodo varias veces), e información del camino más corto para llegar al AgMóvil (descartando los ciclos). Todo lo anterior suponiendo la confiabilidad del camino formado, es decir, que ningún nodo dentro del camino de proxies ha fallado.

Como puede observarse en la Figura 2.7, se muestra la primera forma en la que se aborda el problema de la volatilidad de caminos de proxies grandes, la cual consiste en:

1) La organización de las redes, donde sólo se tiene un único lugar de entrada y salida de cada zona.

2) Que haya un Agente que esté asociado al AgMóvil y que administre la información sobre el último sitio conocido de dicho agente (para su localización). Además todo camino para localizar a un agente comienza en el nodo del AM y sigue en el último sitio que el AM conoce que el AgMóvil estaba en ejecución.

3) Recortar los caminos forzando a que el AgMóvil pase por el nodo *Gateway* al abandonar una red (ver Figura 2.8), actualizando la información en el AM.

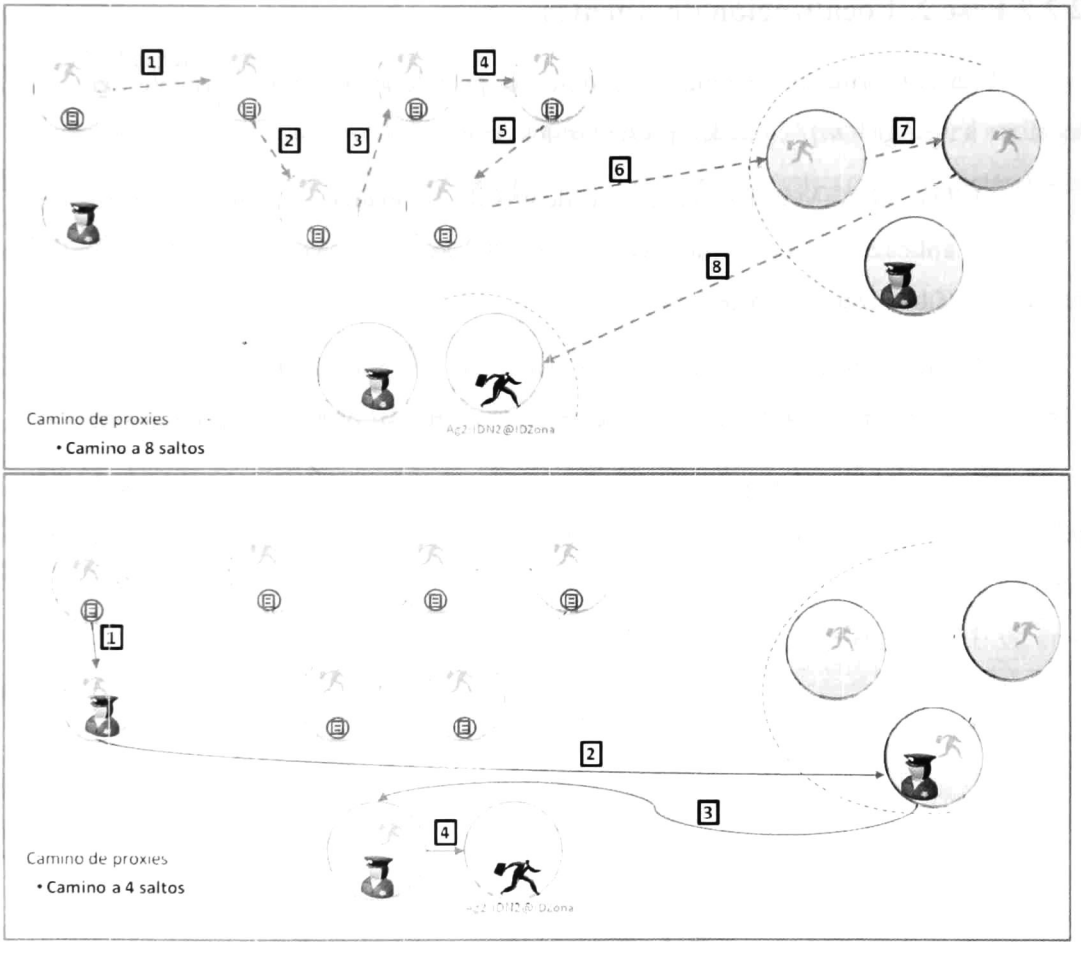


Figura 2.8 Disminución del camino de proxies formado, al asociar un AgMóvil a un AM.

Sin embargo, esta estrategia solo funciona cuando el AgMóvil migra en varios sitios de organizaciones diferentes. Para recortar el camino de proxies cuando el agente migra en varios sitios de la misma zona, se sigue la estrategia de la actualización del valor *LastKnowingLoc* que maneja el AM en periodos determinados de tiempo. En esta estrategia, cada cierto tiempo el AgMóvil se comunica con el AM de la zona local para actualizar su ubicación y así disminuir el camino de proxies formado. Esta estrategia se explicará en seguida con más detalle.

2.2.2 Fase 2. Localización de Agentes

Una vez formado el camino de proxies, es posible localizar a un Agente móvil. Para localizar a un AgMóvil se pueden presentar dos casos:

1) Que el AgMóvil se encuentre dentro de la misma red donde se encuentra su aplicación propietaria (ver Figura 2.9), es decir, este dentro de su red local (Localización Local).

2) Que el AgMóvil haya migrado a una red distinta (ver Figura 2.10) a donde se encuentra su aplicación propietaria y por tanto su AM (Localización externa).

Para una mejor comprensión del protocolo propuesto se presentarán los dos casos por separado, considerando que el protocolo de localización resulta de la fusión de ambos casos.

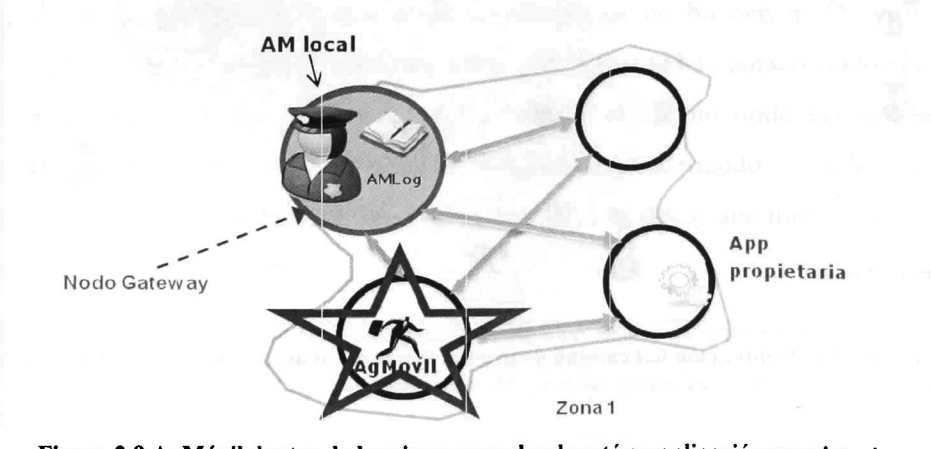


Figura 2.9 AgMóvil dentro de la misma zona donde está su aplicación propietaria

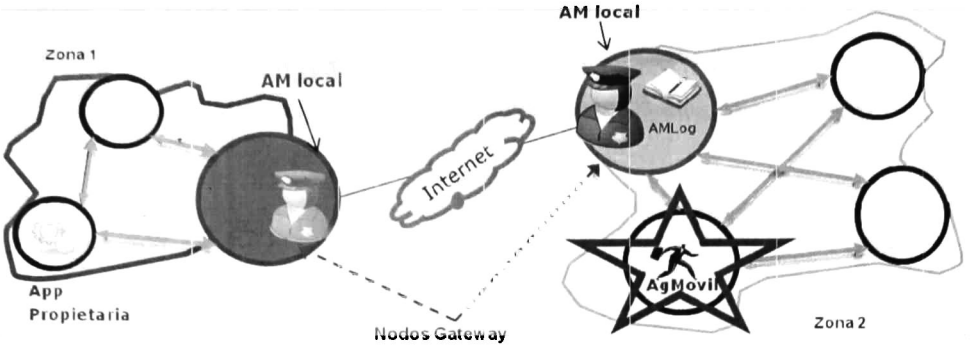


Figura 2.10 AgMóvil dentro de una zona distinta a donde se encuentra su aplicación propietaria

2.2.2.1 Localización Local

Localización local se refiere a que el agente se encuentra dentro de la red en la que fue creado y por tanto, en la misma red donde se encuentra su Aplicación propietaria y su AM propietario.

Para iniciar el proceso de Localización de Agentes, se supone que se ha formado un camino de proxies con información tanto en el AM como en los AMLogs visitados.

El Agente encargado de iniciar el proceso de Localización de Agentes es el AM a quien se asoció el AgMóvil cuando inició ejecución dentro de su red. Como ya se mencionó, el camino de proxies comienza en este nodo y de aquí el primer salto es el último nodo que el AM conoce donde el AgMóvil estaba ejecutándose.

A continuación se describen las operaciones que realizan el AM y el AMLog en el proceso de localización de Agentes dentro de una red local.

Algoritmo localización_AgMóvil() del AM

1. Recibir un mensaje *Loc_Request* de una AppOwner para localizar a un AgMóvil.
2. Buscar en su tabla de agentes el AgMóvil
3. Obtener de la tabla de agentes IDNode@IDZona del campo *LastKnowingLoc*
4. Si IDNode==My.IDNode entonces // AgMóvil está en el nodo actual,
   Enviar MSG("*Inform_current_Loc*", AppOwner);.
   Terminar
   Fin_si

Sino

Si el IDZona==My.IDZona entonces //(red actual) */localización local*

Iniciar *localizaciónLocal(AgMóvil, LastKnowingLoc);*

Sino

Iniciar *localizaciónExterna(AgMóvil, LastKnowingLoc); // descrita en sig. sección*

5. Terminar.

Función localizaciónLocal(AgMóvil, LastKnowingLoc){

Hacer IDNode@IDZona= LastKnowingLoc

Enviar MSG ("*is_in_my_Zone*", *AppOwner)*

Hacer

Enviar MSG("*Req_Where_is*", AMLog, IDNode@IDZona)

Si recibe mensaje *Redirect*, entonces

Actualizar IDNode@IDZona con información del mensaje recibido.

Mientras tiempo_espera_max>0 y no se reciba mensaje *Inform_Here_is*

Si tiempo_espera_max==0, entonces

Enviar MSG ("*Inform_Missing*", AppOwner) // informar que el AgMóvil no fue encontrado

Eliminar referencias del agente *//o iniciar el proceso de recuperación_AgMóvil().*

fin_si

Si se recibió *Inform_Here_is*, entonces

Enviar MSG("*inform_current_Loc*", AppOwner)

Actualizar referencias al AgMóvil.

Fin_si }

Hilo de localización () para AMLog

1. Recibir un mensaje *Request_Where_is*
2. Buscar en su tabla de agentes el AgMóvil
3. Obtener de la tabla de agentes IDNode@IDZona del campo *LastKnowingLoc*
4. Si el IDNode==My.IDNode (*current*), entonces

   Enviar MSG("*Inform_Here_is*")

   Sino enviar MSG("*Redirect*", AM)

   enviar MSG(" *Fwd:Request_Where_is*", AMLog, IDNode@IDZona)
5. Terminar

En la Figura 2.11 puede observarse el intercambio de mensajes entre los elementos del protocolo para el proceso de localización de agentes, cuando el agente se encuentra a un salto del camino de proxies. Se muestran los siguientes pasos:

1. La *AppOwner* envía un *Loc_Request* al AM local

2. El AM recibe el mensaje y responde a la *AppOwner* con un *Is_in_my_Zone* indicando que el agente se encuentra en la red local;

3. El AM envía al AMLog del N_1 (último sitio conocido) un mensaje con un *Request_Where_is*;

4. El AMLog (como el AgMóvil buscado está actualmente en el nodo), responde con un *Here*_is; y

5. El AM de la zona envía un *Inform_current_loc* a la *AppOwner* para informar de la ubicación del AgMóvil buscado.

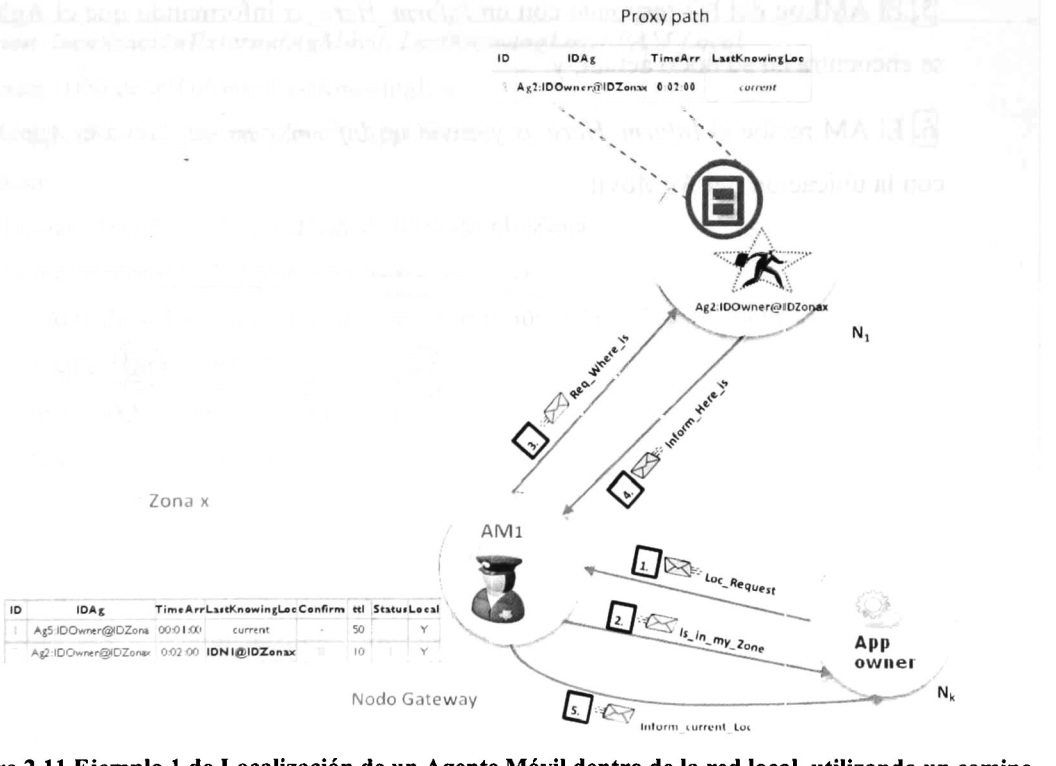


Figura 2.11 Ejemplo 1 de Localización de un Agente Móvil dentro de la red local, utilizando un camino de proxies

En la Figura 2.12 se muestra otro caso de localización local de un AgMóvil cuando el AgMóvil se encuentra a más de un salto en el camino de proxies. Los pasos que se siguen son:

1. La *AppOwner* envía un *Loc_Request* al AM local;

2. El AM recibe el mensaje y responde a la aplicación con un *Is_in_my_Zone* indicando que el agente se encuentra en la red local;

3. El AM envía al AMLog del N_1 (último sitio conocido) un mensaje con un *Request_Where_is;*

4. El AMLog del N_1 (como el agente migró a otro sitio), envía un mensaje *Redirect* para indicarle al AM que el AgMóvil ya no se encuentra en el nodo y que migró a otro nodo, además quiere decir que el AMLog reenviará el mensaje *Request_Where_is* al siguiente sitio;

5. El AMLog del N2 responde con un *Inform_Here_is* informando que el AgMóvil se encuentra en su nodo actual; y

6. El AM recibe el *Inform_Here_is* y envía un *Inform_current_Loc* a la *AppOwner* con la ubicación del AgMóvil.

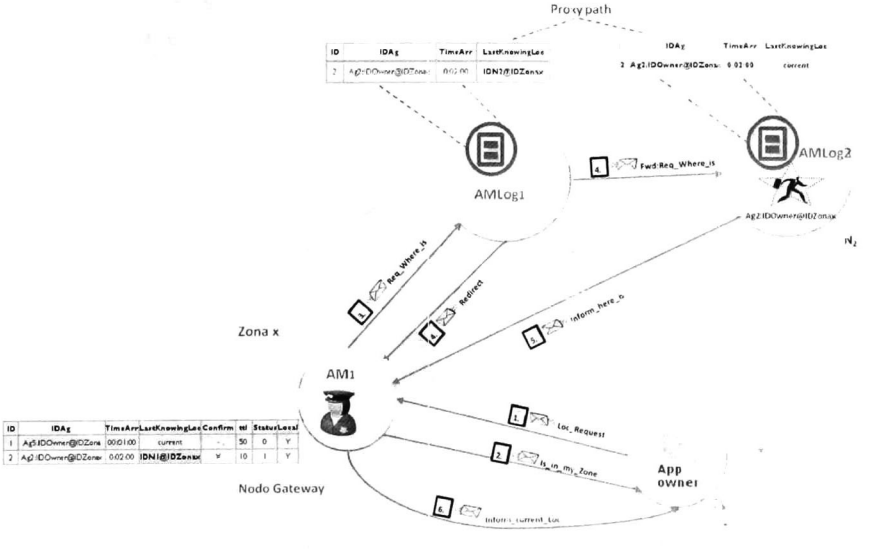


Figura 2.12 Ejemplo 2 de Localización de un Agente Móvil dentro de la red local, utilizando un camino de proxies

2.2.2.2 Localización Externa

La localización externa se refiere a que el agente se encuentra fuera de la red en la que fue creado y por tanto, en una zona distinta a donde se encuentra su Aplicación propietaria y su AM propietario.

Para iniciar el proceso de Localización de Agentes, se supone que se ha formado un camino de proxies con información tanto los AMs como en los AMLogs visitados.

El Agente encargado de iniciar el proceso de Localización de Agentes es el AM propietario, es decir, a quien se asoció el AgMóvil cuando inició ejecución en el sistema. Como ya se mencionó, el camino de proxies comienza en el nodo del AM propietario (de la red donde se encuentra la *AppOwner*) y de aquí el primer salto es el último nodo que el AM conoce donde el AgMóvil estaba ejecutándose.

A continuación se describen las operaciones que realizan los AMs y los AMLogs en el proceso de localización Externa de Agentes.

```
Función localizaciónExterna(AgMóvil, LastKnowingLoc){ //AM Local
  Hacer IDNode@IDZona=LastKnowingLoc
  Enviar MSG("is_out_my_Zone", AppOwner)
  Hacer
    Enviar MSG("Loc_Request", AM, IDNode@IDZona
    Si recibe mensaje Redirect, entonces
        Actualizar IDNode@IDZona con información del mensaje recibido.
    Si recibe mensaje Inform_is_in_my_Zone entonces
      incrementar(tiempo_espera_max)
      Actualizar IDNode@IDZona
    Fin_si
  Mientras tiempo_espera_max>0 y no se reciba mensaje Inform_current _Loc
  Si tiempo_espera_max==0, entonces,
    Enviar MSG("Inform_missing", AppOwner) // AgMóvil no fue encontrado
    Eliminar referencias del agente    //(o iniciar el proceso de recuperación_AgMóvil()).
    Fin_si
  Si se recibió Inform_current_Loc, entonces,
    Enviar MSG("Inform_current_Loc" AppOwner)
```

```
    Actualizar referencias al AgMóvil.
   Fin_si }
```

Hilo de LocalizaciónExterna para el AM de otra zona

```
Recibir un mensaje Loc_Request para localizar a un AgMóvil.
Buscar en su tabla de agentes el AgMóvil
Obtener de la tabla de agentes IDNode@IDZona del campo LastKnowingLoc
Si IDNode==My.IDNode entonces  //el AgMóvil está en el nodo actual)
   Enviar MSG("Inform_current_Loc", AM),
   Terminar.
   Fin_si
 Si_no
    Si el IDZona ==My.IDZona entonces // (red local)
      Enviar MSG ("Inform_is_in_my_Zone", AM)
       IDNodo@IDZona = localizaciónLocal(AgMóvil, LastKnowingLoc)
       Enviar MSG("Inform_current_Loc"+ IDNodo@IDZona, AM)
       Terminar
      Fin_si
      Si_no
       Si el IDZona!=My.IDZona  // (red distinta) // localización externa
            Enviar MSG ("Redirect"  AM) // al AM propietario del AgMóvil
            Enviar un MSG("Fwd:Loc_Request"+AgMóvil, AM, IDNodo@IDZona)
         Fin_si
Terminar
```

Hilo de localización () para AMLog

```
1. Recibir un mensaje Request_Where_is
2. Buscar en su tabla de agentes el AgMóvil
3. Obtener de la tabla de agentes IDNode@IDZona del campo LastKnowingLoc
4.  Si el IDNode==My.IDNode (current), entonces
        Enviar MSG("Inform_Here_is")
    Sino enviar MSG("Redirect", AM)
        enviar MSG(" Fwd:Request_Where_is ",  AMLog, IDNode@IDZona)
5. Terminar
```

En la Figura 2.13 se muestra un caso de Localización externa donde un AgMóvil ha migrado a la red de una organización distinta a donde está la aplicación que lo inició (*AppOwner*). En este caso se siguen los pasos:

[1] La *AppOwner* envía un *Loc_Request* al AM de la red local;

[2] El AM determina que el AgMóvil está en una red distinta y envía un *is_out_my_Zone* a la *AppOwner*.

[3] El AM envía un *Loc_Request* al AM del nodo según la información del registro *LastKnowingLoc*.

[4] El AM externo recibe el mensaje y determina que el AgMóvil está dentro de su zona, por lo que contesta con un *is_in_my_Zone*.

[5] El AM externo envía un mensaje de *Req_Where_is* al AMLog del nodo en *LastKnowingLoc*

[6] El AMLog revisa su tabla de arribos y determina que el AgMóvil está en el nodo actual por lo que envía un mensaje de *Inform_Here_is* a su AM local

[7] El AM recibe el mensaje de su AMLog con la ubicación del AgMóvil y le envía un mensaje de *Inform_current_Loc* al AM propietario del AgMóvil con la información necesaria para su localización.

[8] El AM propietario recibe la ubicación del AgMóvil y se lo informa a la *AppOwner* con un mensaje de *Inform_current_Loc*.

En las figuras Figura 2.14 y Figura 2.15, se muestran otros dos casos de Localización de un AgMóvil que ha migrado a varios nodos de distintas redes del WF-Interorganizacional; además se muestran, numerados, los mensajes que intercambian los elementos del protocolo hasta localizar a un AgMóvil.

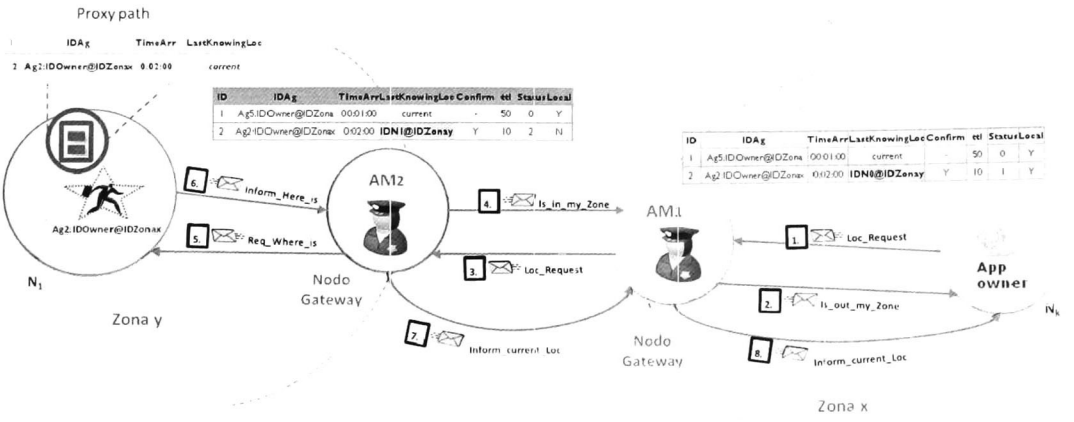


Figura 2.13 Ejemplo 3 de Localización de un AgMóvil en una red distinta a donde fue iniciado

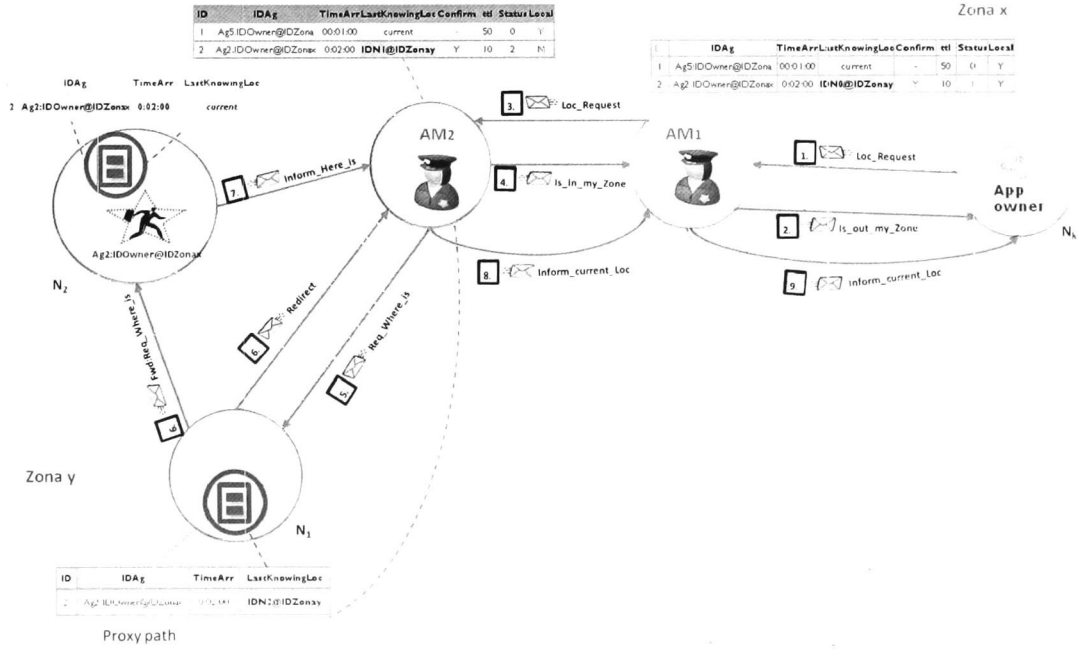


Figura 2.14 Ejemplo 4 de localización de un AgMóvil, que ha migrado a tres sitios de una red distinta a donde fue iniciado.

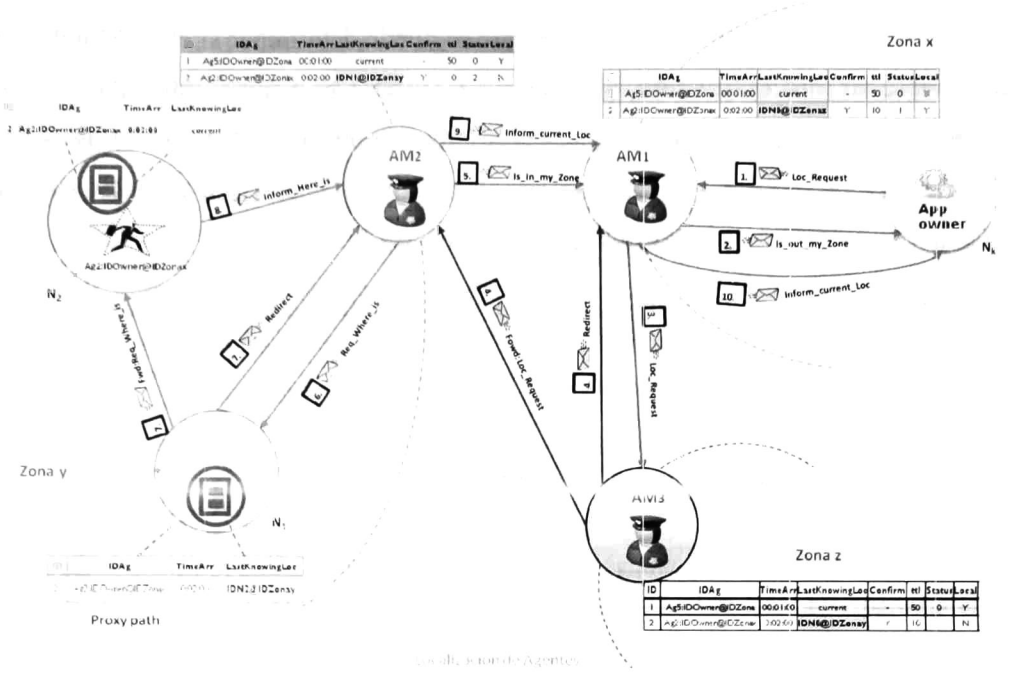


Figura 2.15 Ejemplo 5 de Localización de un AgMóvil que ha migrado por distintas redes del WF-Interorganizacional.

2.3 Protocolo de Control de Agentes Huérfanos

El control de agentes huérfanos consiste en detectar cuándo la aplicación que inició al AgMóvil ya no está disponible haciendo la ejecución de este último innecesaria (ver Figura 1.12).

El concepto más comúnmente utilizado para soportar la *detección de agentes huérfanos* es el basado en *energía*. La *energía* es un tiempo limitado (permiso de ejecución, tiempo de vida) que se asigna a cada AgMóvil durante su creación para consumir con cada acción que realiza; así cuando este tiempo es consumido y el agente no ha terminado las tareas para las cuales fue creado, el agente tendrá que solicitar más tiempo a su aplicación propietaria para continuar con su ejecución ((Baumann, 1999; Jochum, 1997).

Con este concepto, se evita la existencia de agentes que estén en ejecución de manera innecesaria, o se bloqueen al no poder regresar resultados a su aplicación

propietaria. Además se tiene la ventaja de que se puede determinar si el AgMóvil continúa ejecutándose de manera exitosa, ya que si se termina el tiempo asignado y no se ha comunicado con su manejador, entonces podría determinarse si el AgMóvil ha sufrido de alguna falla y por lo tanto, evitar que la aplicación propietaria se bloquee en espera de resultados de un AgMóvil que ya no se está ejecutando.

En esta tesis este concepto se adapta para trabajar con problemas de WF-Interorganizacional.

2.3.1 Gestión del concepto de Energía

Normalmente quien asigna *energía (time to live, ttl)* a un AgMóvil es su aplicación propietaria, en esta tesis se ha delegado ese control al AM de la zona en la que el AgMóvil es creado. El AM propietario tiene la función de administrar el tiempo asignado al AgMóvil para que se ejecute sin comunicarse con su manejador. Una vez que el tiempo es cumplido, el AgMóvil tendrá que comunicarse con su manejador (AM) para *renovar* el permiso de ejecución. Si pasado este tiempo, el manejador determina que la ejecución del AgMóvil es obsoleta o innecesaria, entonces simplemente no renovará este permiso, por lo que el AgMóvil tendrá que terminar con su ejecución liberando así los recursos que se encuentre utilizando; pero si la ejecución aún es necesaria (y su aplicación propietaria sigue activa), entonces este permiso se renovará (Ver Figura 2.16).

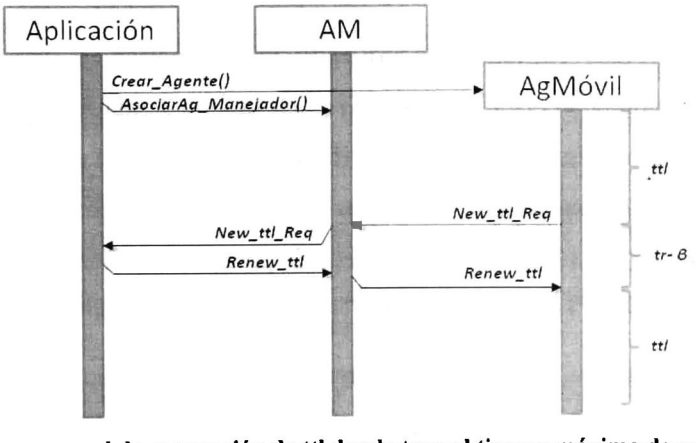


Figura 2.16 Proceso general de renovación de ttl donde tr es el tiempo máximo de espera del AgMóvil para renovar su permiso de ejecución

Para la administración de este *ttl*, el AM agregará en su tabla de agentes (ver Figura 2.3) el tiempo de vida (permiso de ejecución, tiempo permitido que tendrá el AgMóvil para ejecutarse sin comunicarse, etc) asignado. Pueden presentarse dos casos, 1) que el AgMóvil inicie el proceso de renovación de *ttl* cuando se encuentra dentro de la zona donde se encuentra su aplicación propietaria, o 2) que el agente inicie el proceso de renovación de *ttl* estando en la red de una zona distinta a donde se encuentra su aplicación propietaria. Ambos casos son administrados por los AM de la zona en la que el AgMóvil se encuentra. A continuación se explican ambos procesos.

2.3.1.1 Renovación del tiempo de vida de un Agente móvil local

Cuando un AgMóvil se encuentra en ejecución dentro de la misma zona donde fue creado (donde se encuentra su aplicación propietaria), el AM que recibirá la petición será su manejador, es decir el AM a quien fue asociado durante su creación.

Los pasos a seguir para controlar el *ttl* de un AgMóvil que se encuentra dentro de la misma zona donde está su aplicación propietaria son:

Algoritmo para el control de ttl

1. Cuando un AgMóvil es creado, se asocia al AM local y se le asigna un *ttl*
2. Cada AgMóvil tiene un *ttl* que consume cada vez que usa recursos y tiene un tiempo de reserva (*tr*) para soportar proceso de renovación de *ttl*.
3. Cuando en el AgMóvil *ttl* == 0, AgMóvil detiene su ejecución normal e inicia proceso de renovación de *ttl* enviando un mensaje *New_ttl_Req* al AM y espera hasta *tr* por un mensaje de renovación. Si *tr* se cumple, entonces el agente se considera como Agente huérfano y termina su ejecución.

Algoritmo de *renovación_ttl de un AgMóvil*

```
Función renovación_ttl() {
  Si ttl==0 entonces,
  Mientras tr >0
    Enviar MSG ("New_ttl_Req", AM_ local)
```

```
    Esperar te unidades de tiempo.
      Si recibe mensaje Renew_ttl salir
       Si recibe mensaje Reject_ttl salir
  fin_mientras
  Si tr >=0  y mensaje Renew_ttl recibido, entonces
       Renovar ttl con la nueva  cantidad y continuar ejecución
   Si no terminar_ejecución()
Fin_si
```

Algoritmo renovación_*ttl() de un AM local*

```
Función renovación_ttl(){
  Recibir un mensaje New_ttl_Req
 Verificar  en tabla de Agentes que se trata de un AgMóvil local
  Si existe asociación con el AgMóvil entonces
    EnviarMSG(New_ttl_Req , AppOwner)
   Hacer
      Esperar()
   Mientras no se reciba mensaje y te>0
   Si te>=0 y  mensaje Renew_ttl recibido entonces
       Actualizar referencias al AgMóvil
       Enviar MSG(New_ttl, AgMóvil)
     Si_no   eliminar Referencias del AgMóvil
    sino
        iniciar renovaciónExterna_ttl();  // proceso descrito en la siguiente sección
 fin_si

}
```

En la Figura 2.17 y Figura 2.18 se muestran dos casos del proceso de renovación de *ttl* de un AgMóvil. El AgMóvil (que se encuentra en la misma red donde está su aplicación propietaria) es quien inicia el proceso de renovación al enviar un mensaje al AM local.

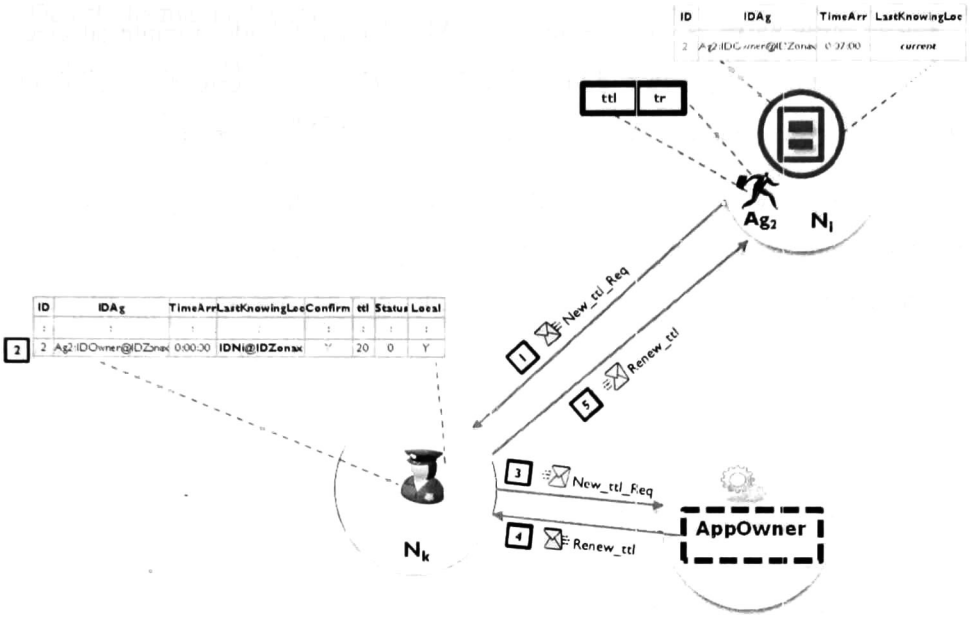


Figura 2.17 Proceso de renovación de ttl iniciada por un AgMóvil dentro la zona donde fue creado, renovación aprobada

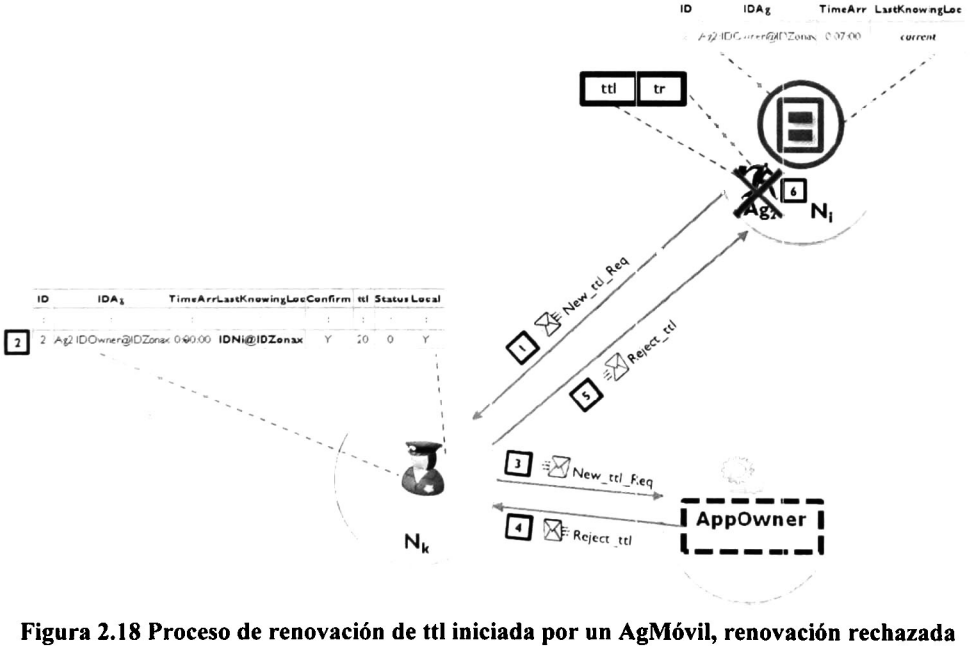


Figura 2.18 Proceso de renovación de ttl iniciada por un AgMóvil, renovación rechazada

Puede presentarse el caso de que el AgMóvil haya fallado durante su ejecución y por lo tanto no inicie el proceso de renovación de *ttl*. Cuando esto ocurre, la aplicación podría quedarse bloqueada esperando por resultados del AgMóvil creado.

Para solucionar lo anterior, el AM propietario registra el *ttl* asociado al AgMóvil durante su creación e inicia un temporizador para esperar, durante *ttl* unidades de tiempo, la llegada de un mensaje de *New_ttl_Req* enviado por el AgMóvil. Si este tiempo ha transcurrido, el AM propietario inicia el proceso de buscar al AgMóvil para renovar su *ttl* (si es que esto es aprobado aún por la Aplicación propietaria).

Hilo *control_ttl_AgMóvil() del AM local*

```
Función control_ttl_AgMóvil(){
//Recibir una referencia a un AgMóvil, registrarlo y asociar un ttl
 Mientras ttl >0
    Esperar por mensaje New_ttl_Req
   Si recibe mensaje New_ttl_Req entonces
       Iniciar  renovación_ttl()
      reiniciar contador
    Fin_mientras
 Si ttl==0  entonces   //no se recibió mensaje de petición de renovación de ttl
  Iniciar localización_AgMóvil()
 Enviar MSG( New_ttl_Req, AppOwner)
 Mientras te<0 hacer
     Si mensajes inform_current_Loc y renew_ttl recibidos salir del mientras
     Si Reject_ttl recibido entonces eliminar referencias al AgMóvil y salir
  Fin_mientras
 Si mensaje inform_current_Loc y renew_ttl recibidos entonces
    Enviar MSG(New_ttl, AgMóvil)
    Actualizar referencias al AgMóvil y reniciar contador de ttl
  Si_no
     Si te==0 y Renew_ttl recibido entonces,
       Eliminar referencias al AgMóvil //o iniciar proceso de Recuperacion_AgMóvil();
  Fin_si
Fin_si}
```

De esta forma, el AM detectaría cuando un AgMóvil ha dejado de ejecutarse debido a alguna falla (aunque también es posible que haya un retardo en la llegada de los mensajes o que la red se haya fraccionado y que el AgMóvil siga en ejecución) y podría iniciar el proceso de recuperar el estado del AgMóvil que falló; también se daría cuenta cuando la aplicación ya no estuviera disponible. Todo lo anterior se detectaría hasta que el *ttl* se haya consumido. Cuando la red es propensa a sufrir grandes retrasos en el envío y recepción de mensajes, el elegir un *ttl* y un tiempo de espera (*te*) adecuados es muy importante.

Cada vez que un AgMóvil renueva su *ttl*, actualiza la información sobre el último sitio donde se ha ejecutado (*LastKnowingLock)* acortando así el camino de proxies creado. Por lo tanto, entre menor sea el *ttl* asignado al AgMóvil se actualizaría más rápido esta información, reduciendo la posibilidad de que el camino de proxies se perdiera debido a la falla de un nodo intermedio. Pero si el *ttl* es muy pequeño, el número de mensajes que se intercambiarían entre los elementos del protocolo, se incrementaría.

Cuando un AgMóvil migra a otra red, el AM local es el encargado de renovar su *ttl* y por lo tanto tendrá que comunicarse con el AM propietario del AgMóvil.

2.3.1.2 Renovación del tiempo de vida de un Agente móvil externo

Cuando un AgMóvil migra a otra red, tiene que registrarse en el nodo *Gateway* de la red a la que migrará. Al entrar a la red, el AM local a esa red lo registra en su tabla de agentes y además le asigna un nuevo tiempo de vida (*ttl*) o permiso de ejecución para que el AgMóvil se ejecute dentro de esta red. Es decir, el AgMóvil tendrá dos *ttls,* uno asignado por su AM propietario y uno asignado por el AM de la red que visita. Pero esta información solo la registra y maneja el AM de la red local y no el AgMóvil.

El AgMóvil solo recibe un tiempo de vida (*ttl)* al entrar a una red y cuando éste es agotado, le solicita al AM local la renovación del mismo. De esta forma, en cada red se podrían utilizar distintos permisos para los AgMóvil (para que se ejecuten sin comunicarse con el AM local), cada uno dependiente de las condiciones de la red de la que se trate. Por lo tanto, cada AM podría decidir hasta cuando seguir concediendo permiso de ejecución a los AgMóviles externos.

Sin embargo, la información del *ttl* original no se pierde; ésta es registrada y administrada por el AM de la red que el AgMóvil está visitando. Cuando el *ttl* original es cumplido, el AM de la red local se comunica con el AM propietario del AgMóvil para renovar su tiempo de vida.

Primero veamos el caso en que el AgMóvil pide tiempo de vida al AM de la red local, es decir, el AgMóvil inicia el proceso de renovación de *ttl* en una red externa. El algoritmo a seguir por el AgMóvil es:

AgMóvil *renovación_ttl()*

```
Función renovación_ttl() {
   Mientras tr >0
    Enviar MSG(New_ttl_Req,  AM local)
   Esperar te unidades de tiempo.
   Si recibe mensaje Renew_ttl  entonces salir
   Si recibe mensaje Reject_ttl entonces salir
  fin_mientras
  Si tr >=0  y Renew_ttl recibido, entonces
        renovar ttl con la nueva  cantidad y continuar ejecución
  Si no terminar_ejecución()
}
```

Como puede observarse, los pasos a seguir por el AgMóvil son los mismos a que si estuviera en la red donde está su aplicación propietaria y su AM propietario. Ahora, los pasos a seguir por el AM local cuando recibe una solicitud de Renovación de *ttl* de un agente externo son:

AM local renovaciónExterna_*ttl()*

```
Function renovaciónExterna_ttl(){  // no existe asociación con el AgMóvil entonces
//AgMóvil externo
    Actualizar referencias al AgMóvil
   Enviar MSG(New_ttl,  AgMóvil) //aprobar renovación
}
```

Podría seguirse otra política de renovación de *ttl* e incluso rechazar la ejecución del AgMóvil, por lo que este último tendría que buscar otro sitio de otra red para continuar su ejecución. En este caso se supondrá que siempre se aprueba la renovación.

Así, cuando un AgMóvil abandone la red que un AM administra, el *ttl* es actualizado por el *ttl original* que le había asignado su AM propietario.

Igual que en el caso de renovación de tiempo de vida de un AgMóvil local, al transcurrir *ttl* unidades de tiempo el AM debe iniciar el proceso de renovación. Cuando se trata de un AgMóvil externo el AM puede iniciar el proceso dos veces: 1) cuando el *ttl* local se ha cumplido y se requiera localizar al AgMóvil de manera local; y 2) cuando el *ttl original* se ha cumplido y deba ser renovando a través del AM propietario.

Para el primer caso se tiene:

AM local *control_ttl_AgMóvilExterno()*

```
Function control_ttl_AgMóvilExterno(){
 Registrar la llegada de un AgMóvil externo y asignarle un ttl local (ttl2).
  Iniciar control_ttl1_AgMóvilExterno(); //mostrado más adelante
  Iniciar control_ttl2_AgMóvilExterno();
}
Función control_ttl2_AgMóvilExterno(){
  Mientras ttl2 >0
    Esperar por mensaje New_ttl_Req
   Si recibe mensaje New_ttl_Req iniciar proceso renovaciónExterna_ttl()
      reiniciar contador
   Fin_mientras
 Si ttl2==0  entonces   //no se recibió solicitud de renovación
  Iniciar localización_AgMóvil()
  Mientras te<0 hacer
     Si inform_current_Loc recibido salir del mientras
  Fin_mientras
  Si inform_current_Loc recibido entonces
    Enviar MSG(New_ttl, AgMóvil)
    Actualizar referencias al AgMóvil
     reniciar contador de ttl2
```

```
  Si_no Si te==0 entonces,
    Eliminar referencias al AgMóvil //o iniciar proceso de Recuperacion_AgMóvil();
    Informar al AM propietario del fallo del AgMóvil
  fin_si
Fin_si
}
```

Como ya se mencionó, en este caso vamos a suponer que el AM de una zona siempre aprueba el *ttl* local asignado a un AgMóvil externo. Ahora, para el segundo caso se tiene:

Control_ttl1_AgMóvilExterno() por el AM Local

```
Función control_ttl1_AgMóvilExterno(){
Mientras ttl1 >0
    Esperar,
Fin_mientras
Si ttl1==0  entonces
  Enviar MSG(New_ttl_Req,  AM propietario) // (manejador del AgMóvil)
  Mientras te<0 hacer
     Si renew_ttl recibido salir del mientras
     Si Reject_ttl recibido entonces
        Eliminar referencias al AgMóvil
        Enviar MSG("termination_req", AgMóvil)// para que termine ejecución y salir
     fin_si
  Fin_mientras
  Si Renew_ttl recibido entonces
    Actualizar el ttl1 en la tabla de agentes
    Reniciar contador de ttl1
  Si_no Si te==0  entonces,
    Eliminar referencias al AgMóvil
    Terminar la ejecución del AgMóvil
  Fin_si
Fin_si
}
```

Al delegar la renovación de *ttl* al AM de la red en la que el AgMóvil se encuentre, se acorta el tiempo en que el AgMóvil está en estado de bloqueo esperando un mensaje de renovación, ya que la renovación se realiza de manera local y el *tiempo de espera* es menor. Por lo tanto, el AgMóvil aumenta el tiempo en que permanece en ejecución pero sin perder la característica de detectar cuándo es un agente huérfano (ver Figura 2.19).

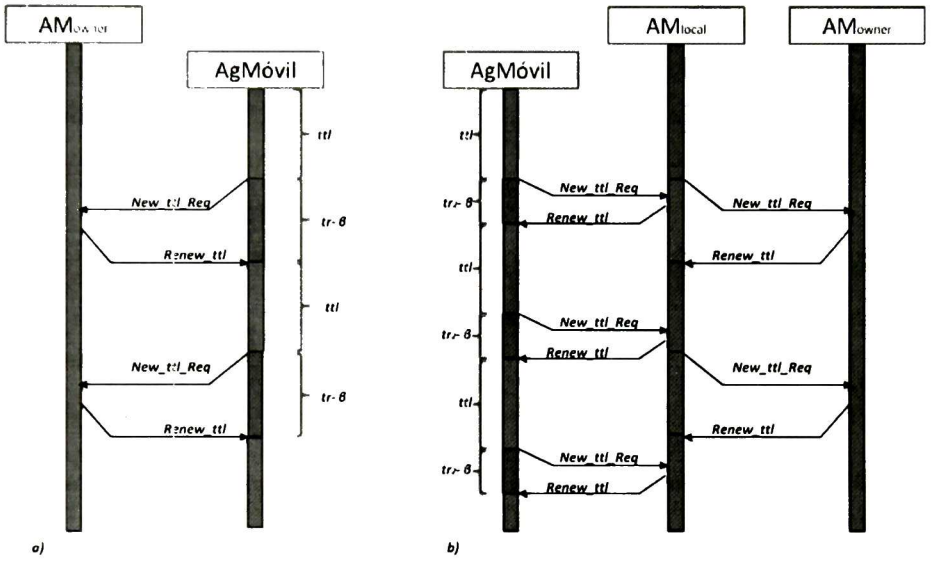


Figura 2.19 Proceso de renovación de *ttl* de un AgMóvil en una red externa, a) renovación con el AM propietario, b) renovación a través del AM local

2.4 Protocolo de Terminación de Agentes

La terminación de Agentes está soportada en dos formas: 1) terminar un AgMóvil cuando determina que es un agente huérfano (eliminación pasiva), y 2) terminar un AgMóvil cuando así lo requiera la aplicación propietaria (eliminación activa).

2.4.1 Eliminación Pasiva de Agentes móviles

La eliminación pasiva de AgMóvil está soportada por el control de agentes huérfano descrito en el protocolo anterior, en donde, cuando ha pasado el tiempo de espera (*te*) para

renovar el tiempo de vida y el AgMóvil no ha recibido un mensaje *Renew_ttl,* entonces éste se declara como agente huérfano y termina su ejecución (ver Figura 2.20).

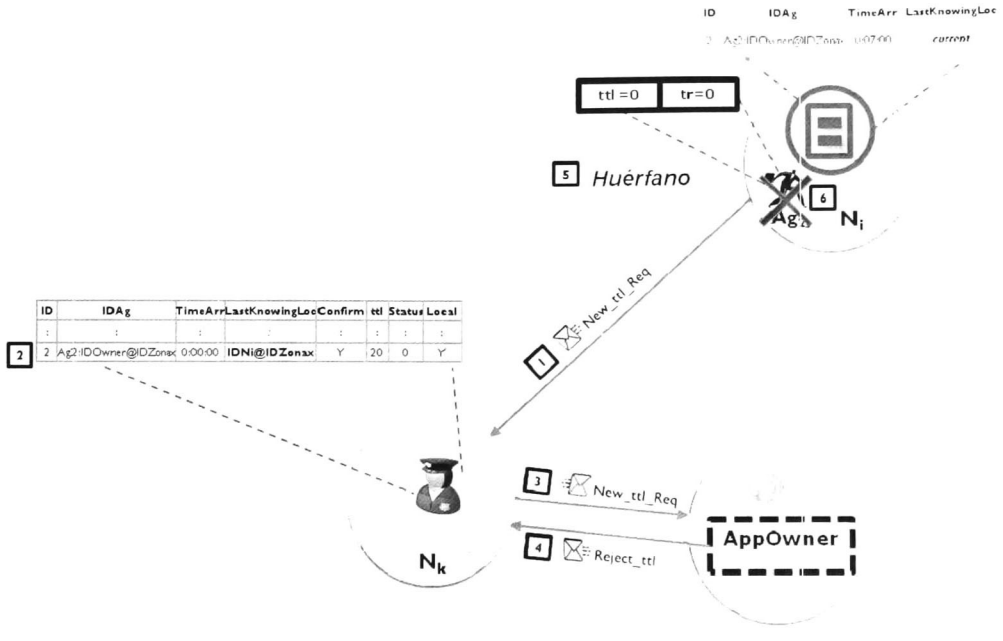


Figura 2.20 Ejemplo de eliminación pasiva de un AgMóvil, transcurridas tr unidades de tiempo

2.4.2 Eliminación Activa de Agentes móviles

La eliminación activa de AgMóvil se refiere a la posibilidad de eliminar a un AgMóvil en un momento determinado, no importando en qué red se esté ejecutando.

La entidad encargada de terminar la ejecución de un AgMóvil es el AM propietario. Cuando un AM recibe la solicitud de terminar la ejecución de un AgMóvil por parte de su aplicación propietaria, éste tiene la tarea de localizar al AgMóvil y enviar un mensaje de terminación. Además, el AM debe eliminar toda referencia al AgMóvil eliminado. Aún si el AgMóvil fallara en recibir un mensaje de terminación (*Termina_request),* cuando su *ttl* se cumpla no podrá ser renovado, pues las referencias al AgMóvil han sido eliminadas. En esta situación el AgMóvil se consideraría agente huérfano y terminaría su ejecución. Al eliminar toda referencia a un AgMóvil se asegura que éste eventualmente termine su ejecución (la cual ocurriría a lo máximo en *ttl+tr* unidades de tiempo).

Como se acaba de mencionar, la localización de un AgMóvil es una parte importante de este protocolo, por lo que se hará uso del camino de proxies generado en el protocolo de localización de agentes ya descrito.

A continuación se describen los pasos que sigue el AM para terminar a un AgMóvil.

AM local *terminacion_AgMóvil()*

```
Función terminación_AgMóvil(){
  Recibir un mensaje Termina_request por parte de una aplicación local.
  Verificar en la tabla de agentes que se trata de un AgMóvil local
 Si existe asociación con el AgMóvil entonces
       Iniciar terminaciónLocal_AgMóvil()
  Sino
      Iniciar terminaciónExterna_AgMóvil(); //descrita más adelante
}
Función terminaciónLocal_AgMóvil(){
     Marcar los registros asociados al AgMóvil para eliminar
     Enviar MSG(ACK_termina, AppOwner)
     Obtener del campo LastKnowinLoc la ubicación del AgMóvil
    Si IDNode@IDZona == IDNodoLocal entonces //AgMóvil en el nodo actual
       EnviarMSG( Termina_request, AgMóvil)
       Eliminar refererencias
        Salir
      Si_no
       Enviar MSG("Fwd:Termina_request", AMLog , IDNode@IDZona)
       Mientras te>0 y ACK_termina no sea recibido, entonces esperar
       Si ACK_termina recibido o te ==0 entonces
        Eliminar referencias
      fin_si
    fin_si
}
```

AM externo *terminacion_AgMóvil()*

```
Función terminaciónExterna_AgMóvil(){
  Marcar los registros asociados al AgMóvil para eliminar
  Enviar MSG(ACK_termina);
  Obtener del campo LastKnowinLoc la ubicación del AgMóvil
  Si IDNode@IDZona ==IDNodoActual entonces //AgMóvil en el nodo actual
     Enviar MSG( Termina_request, AgMóvil)
     Eliminar refererencias
    Salir
   Si_no
     Enviar MSG(Termina_request, AMLog, IDNode@IDZona)
     Mientras te>0 y ACK_termina no sea recibido, entonces esperar
     Si ACK_termina recibido o te ==0 entonces
         Eliminar referencias
 fin_si
fin_si
}
```

AMLog *terminacion_AgMóvil()*

```
Función terminación_AgMóvil(){
   Recibir un mensaje Termina_request.
  Marcar los registros asociados al AgMóvil para eliminar
  Enviar  MSG(ACK_termina ),
  Obtener del campo LastKnowinLoc la ubicación del AgMóvil
  Si IDNode== IDNodeLocal entonces //AgMóvil en el nodo actual
     Enviar MSG(Termina_request, AgMóvil)
     Eliminar refererencias
     Salir
    Si_no
      Enviar MSG(Termina_request, AMLog, IDNode@IDZona)
      Mientras te>0 y ACK_termina no sea recibido, entonces esperar
     Si ACK_termina recibido o te ==0 entonces
       Eliminar referencias
   fin_si }
```

Como puede verse, el mensaje de terminación simplemente se reenvía a través de todos los nodos en el camino de proxies, hasta que el mensaje finalmente llega el nodo donde el AgMóvil se encuentra actualmente. Cuando un AgMóvil recibe un mensaje *Termina_request* simplemente termina con su ejecución. En la Figura 2.21, se muestra un caso de terminación activa donde un AgMóvil se encuentra en una zona distinta a donde fue iniciado, por lo que el mensaje de *Termina_request* es reenviado a los nodos que el AgMóvil ha visitado (a través del camino de proxies) hasta que finalmente llega al sitio donde se está ejecutando.

Los protocolos aquí presentados se utilizan para el control de agentes móviles para realizar tareas de localización de agentes, detección de agentes huérfanos y terminación de agentes. Los protocolos están basados en una distribución de red para aplicaciones de Flujo de trabajo Interorganizacional. Se modelará el comportamiento de estos protocolos en el capítulo 4.

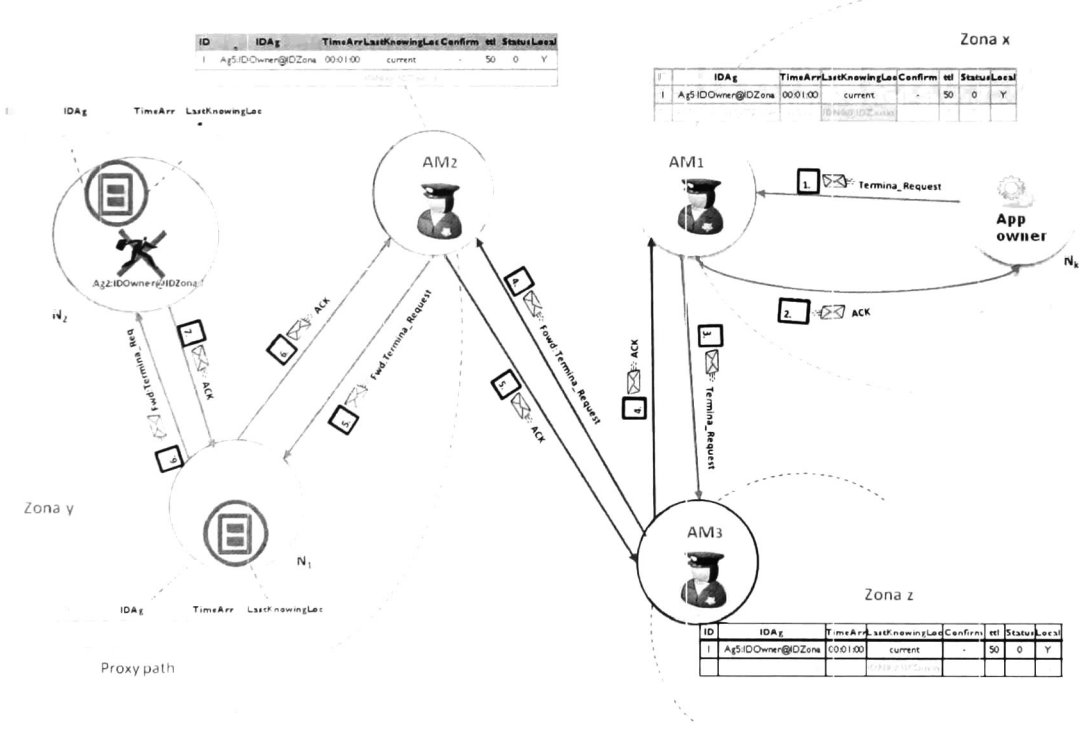


Figura 2.21 Ejemplo de terminación activa de un AgMóvil que se encuentra en una zona distinta a donde está su aplicación propietaria.

Capítulo 3

Protocolo para la recuperación de Agentes Móviles

Resumen. En este capítulo se propone un protocolo para la recuperación de un agente móvil que ha terminado su ejecución debido a la presencia de una falla. La estrategia propuesta está basada en conceptos de agentes testigo y de registro de las operaciones de los agentes utilizando logs.

3.1 Tolerancia a fallas de agentes móviles

La confiabilidad de un sistema depende en gran medida de la tolerancia a fallas, que es la capacidad del sistema o componente de continuar con la operación normal a pesar de la presencia de fallas en software o hardware. En un sistema tolerante a fallas las fallas serán enmascaradas o tratadas apropiadamente.

Debido a que los agentes móviles son programas autónomos capaces de migrar de un sitio a otro en una red de computadoras, éstos son susceptibles a varios tipos de fallas. Cuando una falla ocurre, el agente y/o el sitio donde éste se ejecuta podrían dejar de estar disponibles. Para tratar con estos problemas se necesita asegurar que la tarea a realizar por el agente pueda continuar a pesar de la presencia de una falla. Por esta razón la confiabilidad y la tolerancia a fallas son aspectos muy importantes en el desarrollo de sistemas de agentes móviles.

La replicación es una de las técnicas más usadas para soportar tolerancia a fallas. La idea es replicar varios servidores o replicar a los agentes para enmascarar una falla. De tal forma que, cuando un agente o un servidor terminan con su ejecución debido a la presencia de una falla, aún se pueden obtener resultados de los otros agentes (o servidores). La ventaja principal es que la recuperación es inmediata y no hay bloqueos en la ejecución cuando ocurre una falla.

Sin replicación, un agente móvil es un solo punto de falla y su estado interno puede perderse debido a diversos eventos: la falla del sitio donde se estaba ejecutando el agente, la pérdida del agente durante la migración de un sitio a otro debido a la pérdida de un enlace, la falla en la ejecución del agente móvil dentro de un sitio que continúa disponible, etc.

Sin embargo, el esquema de replicación es muy costoso ya que se tienen que mantener muchos servidores pero con el mismo estado (o múltiples copias de un mismo agente con un estado consistente). Por otro lado, la ejecución de un agente móvil en otro sitio puede resultar en un estado diferente (producir resultados diferentes), por lo que se requiere la implementación de tareas para mantener y preservar la consistencia entre las réplicas. Además las aplicaciones para los que se podría usar este esquema no proporcionarían la propiedad de *exactamente-una-ejecución* de los agentes móviles.

El protocolo presentado en este capítulo se enfoca en la detección de la falla de un agente móvil y en la recuperación de su estado tratando de respetar lo más posible la propiedad *exactamente-una-ejecución* de los agentes móviles. Este protocolo está basado en los conceptos de *agentes testigo* y de registro de las operaciones de los agentes utilizando *logs* y supone que las fallas presentadas son originadas en el software y no en hardware, es decir que los sitios donde se encuentran los agentes siempre están disponibles.

3.2 Protocolo de recuperación de agentes móviles

El protocolo de recuperación de agentes está basado en la integración de cuatro módulos principales (ver Figura 3.1):

Módulo 1) ***Recolección y almacenamiento de datos***. En este módulo se capturan y almacenan datos sobre la ejecución del agente móvil (*LOGs* y *checkpoints*) en cada sitio que visita, los cuales se utilizarán (en el módulo 4) para recuperar su estado en caso de la presencia de una falla.

Módulo 2) ***Monitoreo de agentes y control de testigos***. En este módulo se crea una entidad (*agente testigo*) que se encargará de monitorear si el agente móvil continúa en ejecución o no.

Módulo 3) ***Detección de fallas***. Este módulo está ligado al módulo anterior donde se lleva a cabo el monitoreo del agente móvil; una vez que se detecta un problema este módulo se encarga de especificar qué tipo de falla se presentó, la cual será tratada apropiadamente en el módulo 4.

Módulo 4) ***Recuperación de fallas***. A partir de la detección de una falla, este módulo se encarga de recuperar el estado del agente utilizando la información almacenada en el módulo 1.

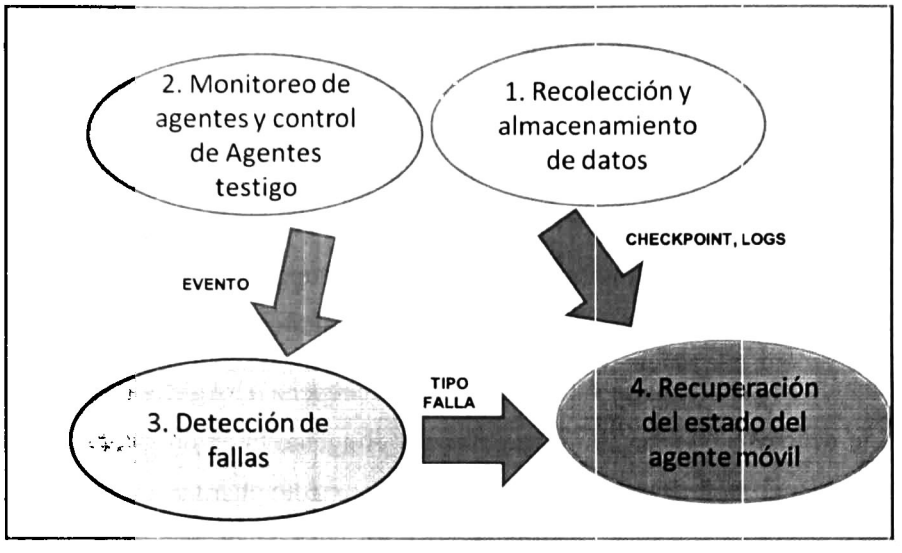


Figura 3.1 Estructura general del protocolo para la recuperación de agentes móviles

Para poder detectar la falla de un agente móvil (AgMóvil, por sus siglas en inglés-Mobile Agent), así como para recuperarlo, se utilizan los conceptos de *logs, checkpoints* y *agentes testigo*. Primero se registran (por cada sitio) todas las operaciones que el agente móvil realiza (*logs*) y se guarda una copia de su estado (*checkpoint*) cuando el agente está listo para migrar a otro sitio (en el módulo 1). Después se crea un agente que monitorea si el AgMóvil está vivo o no (módulo 2); si durante el monitoreo ocurre una falla, se trata de determinar qué tipo de falla fue la que ocurrió (modulo 3), para finalmente tratar la falla de la manera adecuada (módulo 4), ya sea que se recupere el estado del agente móvil (con la información almacenada en los *checkpoints* y en los *logs*) o simplemente se notifique de la falla al agente administrador de la red.

Las entidades principales involucradas en este protocolo son: el Agente Administrador (AM) de la red, el Agente móvil (AgMóvil), los *agentes testigo* y el AMLog de cada sitio. A continuación se detallará el funcionamiento de cada módulo.

3.2.1 Recolección y almacenamiento de datos para el protocolo de recuperación de agentes

En cada sitio visitado por el AgMóvil, el AMLog tiene que registrar (*Log*) cada operación realizada por el AgMóvil mientras se ejecuta en ese sitio. Esta información es de gran importancia para la detección de la falla y la recuperación del estado del AgMóvil. Este *Log* se supondrá que se encuentra en almacenamiento estable.

Cuando se presenta una falla, se tiene que recuperar el AgMóvil que se ha perdido debido a la ocurrencia de la falla. Sin embargo, el agente tiene un estado interno el cual puede perderse al ocurrir la falla. Para ello, se puede guardar el estado del agente (*checkpoint*) cada vez que haya completado las tareas necesarias en un sitio y antes de que se disponga a migrar al sitio siguiente durante la ejecución del caso de flujo de trabajo. De tal forma que si el AgMóvil falla antes de iniciar su ejecución en un sitio, aún podría recuperarse su estado con la ayuda del *checkpoint* almacenado en el sitio anterior. La información del estado del agente también se supondrá que se encuentra en almacenamiento estable (o permanente). En la Figura 3.2, se presentan los elementos principales utilizados en este módulo.

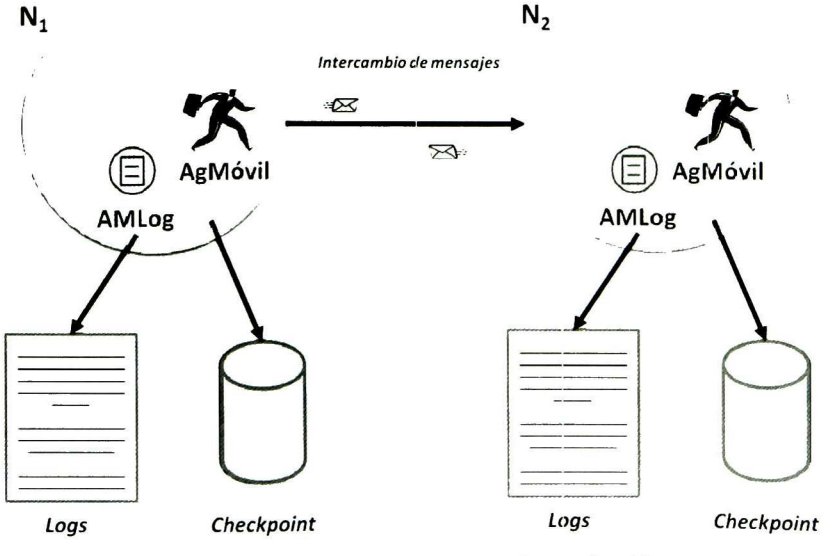


Figura 3.2 Elementos generales en cada nodo utilizados para la recolección y almacenamiento de datos del protocolo de recuperación de agentes

El comportamiento del AgMóvil en este módulo se describe a continuación. Para lo cual se supone que el AgMóvil se encuentra en un sitio N_i y su *agente testigo* se encuentra en el sitio anteriormente visitado N_{i-1}.

Algoritmo general que sigue el AgMóvil durante su ejecución en un sitio Ni.

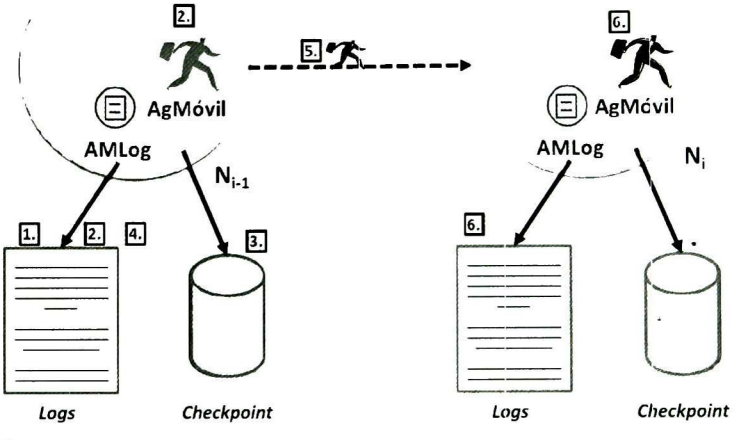
1. *Registrar su entrada al sitio Ni,* LOG(this.getID(), entrada)
2. Mientras *tenga_operaciones_por_realizar_en_sitio() hacer*
   Obtener la siguiente operación a realizar, Operación=getNextOp();
   Realizar la operación, Realizar (Operación);
   Registrar en el log la operación realizada, LOG (this.getID(),Operación);
   Fin_mientras
3. *intentar {*
   Guardar copia de su estado,CHECKPOINT (this.getID(), IDChP, estado),
   LOG(this.getID(),IDChP);
   }capturar(error){ notificar_error(); }
4. *Registrar su salida del sitio Ni, LOG(this.getID(), salida).*
5. *Migrar al sitio siguiente*

Los pasos de este algoritmo se integran en la Figura 3.3. De acuerdo al procedimiento anterior y al esquema de la Figura 3.3, una vez que el AgMóvil llega a un nuevo sitio (N_i), inmediatamente registra su entrada o arribo en el *LOG* del sitio actual en almacenamiento confiable (1.). El propósito de registrar su entrada es para que los agentes testigo sepan que el agente móvil llegó con éxito al sitio. Una vez en el sitio, el AgMóvil realiza las operaciones para las cuales fue diseñado, las cuales serán registradas en el archivo *LOG* (2.), el cual administra el **AMLog** del sitio actual.

Cuando el AgMóvil termina sus tareas, inmediatamente guarda (*checkpoint*) una copia de su estado en almacenamiento confiable (3.). Esta acción también se toma como parte de las tareas del AgMóvil, la cual si no se realiza apropiadamente, genera un error de inconsistencia y la computación del agente no podría continuar. Ya que este estado se

utiliza para la recuperación del AgMóvil es importante que se realice de manera correcta. Después, el AgMóvil registra su *salida* del sitio actual (4.) y deja información del *sitio siguiente* (N_{i+1}) a donde migrará, para finalmente desplazarse a dicho sitio.

Así, toda operación que realiza el AgMóvil en un nodo comienza por el *registro de su entrada* y termina con el *registro de su salida*, terminando así la ejecución del AgMóvil en ese sitio.



1. AgMóvil *registra su entrada*
2. AgMóvil *realiza su computación y guardar LOG de cada operación que realiza*
3. AgMóvil *registra una copia de su estado en el checkpoint*
4. AgMóvil *registra su salida*
5. AgMóvil *migra al sitio siguiente*
6. AgMóvil *llega al sitio siguiente y registra su entrada*

Figura 3.3 Pasos que sigue un Agente Móvil a su arribo en un sitio en el protocolo de recuperación de agentes

3.2.2 Monitoreo de Agentes

Para poder detectar fallas en un agente móvil (AgMóvil) así como para recuperar al agente que falló, se utiliza el concepto de *agentes testigo*. Un agente testigo es un tipo especial de agente que el AgMóvil deja en cada sitio cuando se dispone a migrar al sitio siguiente dentro de su itinerario. Este *agente testigo* es el responsable de monitorear la ejecución del agente actual, para lo cual emplea el envío de mensajes *"Is_Alive?"* cada cierto periodo de tiempo al sitio donde se encuentra el agente móvil y espera por la respuesta del mensaje (*"Im_Alive()"*). Mientras continúe recibiendo respuesta a este

mensaje, el *agente testigo* determina que el agente móvil continúa con su ejecución. Si la respuesta no se recibe, entonces existe la posibilidad de que alguna falla en la ejecución del AgMóvil se haya presentado.

3.2.2.1 Creación de Agentes testigo

Suponiendo que se tiene un agente móvil en ejecución en un sitio N_{i-1}. Cuando este AgMóvil ha terminado su ejecución en N_{i-1}, crea un nuevo agente testigo $Testigo_{i-1}$, para que monitoree su estado mientras éste se ejecuta en el sitio siguiente (N_i) y finalmente el AgMóvil migra al sitio N_i. Esta operación la realiza en cada nodo visitado.

Una vez que el *agente testigo* ($Testigo_{i-1}$) es creado, espera un intervalo de tiempo y comienza a monitorear la disponibilidad del AgMóvil en el sitio siguiente (N_i). El monitoreo, como ya se mencionó, consiste en enviar cada cierto intervalo de tiempo, un mensaje *Is_Alive?* al sitio siguiente para determinar si el agente móvil sigue disponible o no. Si recibe un mensaje *Im_Alive()* del sitio siguiente, esperará otro intervalo de tiempo antes de mandar un nuevo mensaje de monitoreo. Por lo tanto, si se agrega este comportamiento de un *testigo* a la Figura 3.3, se obtendría lo descrito por la Figura 3.4.

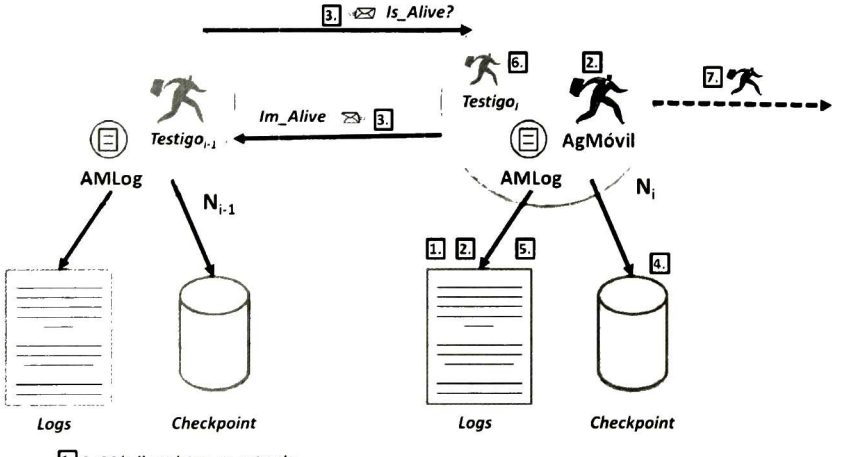


Figura 3.4 Creación de Agentes Testigo

Como puede observarse en la Figura 3.4, una vez que el AgMóvil ha creado un nuevo *agente testigo (Testigo$_i$)* en el Sitio N_i y migrado al sitio siguiente N_{i+1}, el *Testigo$_{i-1}$* en el sitio N_{i-1} que monitoreaba su disponibilidad ya no será el testigo de dicho AgMóvil puesto que ya ha migrado a un nuevo sitio. Sin embargo, el agente testigo anterior (*Testigo$_{i-1}$*) ahora se encargará de monitorear la disponibilidad del nuevo testigo creado (*Testigo$_i$*), es decir al testigo creado más recientemente que monitorea al AgMóvil desde un sitio anterior al sitio actual. Por lo tanto, testigos que fueron creados primero, monitorearán a los agentes testigo que están justo un sitio más cercano al sitio donde se encuentra el AgMóvil (ver Figura 3.5), con lo cual se generará una cadena de monitoreo.

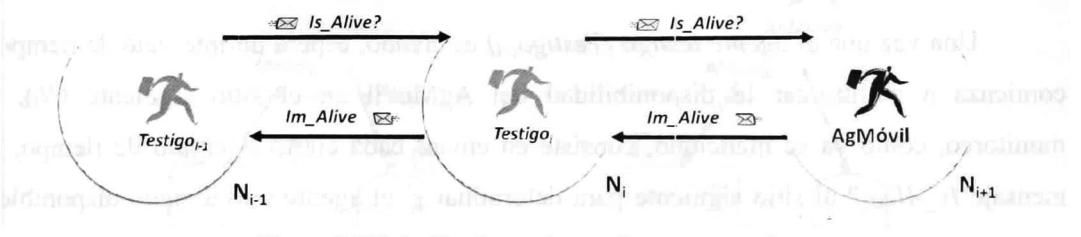


Figura 3.5 Relación de monitoreo de agentes testigo

Las acciones a realizar por cada *Testigo$_i$* se describen a continuación

Algoritmo de Monitoreo() de Agentes Testigo

```
1. Esperar (intervalo_de_ tiempo);
2. Enviar MSG("Is_Alive?" Sitio_Sig);
3. Mientras ( tiempo_espera_max>0 && no_reciba_respuesta())
      Decrementar (Tiempo_espera_maximo);
4. Fin_mientras;
5. Si ( Tiempo_espera_max==0) //no se recibió respuesta en un tiempo adecuado
      Si (el agente a monitorear es el AgMóvil)
         detección_falla(); //módulo 3 de detección de fallas
      Si no
         recuperar_AgTestigo();
   Sino ir_a_paso_1;
6. Fin.
```

Los agentes testigo nos ayudan a monitorear la disponibilidad del AgMóvil en el sitio siguiente, de tal forma que cuando el AgMóvil falla, el testigo en el sitio anterior puede detectar la falla y puede iniciarse el proceso de recuperación del estado del AgMóvil. Pero si ocurre una falla en el agente testigo, no existiría una entidad que detecte la falla del AgMóvil. Es por eso que es necesario tener un testigo que pueda recuperar al *agente* que monitorea al AgMóvil, así si el *testigo* más cercano al AgMóvil falla, la falla puede ser detectada por el *testigo* en el sitio anterior, el cual se encargará de recuperar al agente que ha fallado permitiendo que el monitoreo al AgMóvil continúe. En este módulo preserva la cadena de testigos, con el fin de recuperar a un agente testigo que ha fallado. Las fallas en la ejecución del Agente móvil se describirán en el módulo de detección de fallas.

3.2.2.2 Fallas en los agentes testigo

Antes de que un agente móvil complete sus tareas dentro de los sitios de una zona (o red de una compañía), existe un camino de testigos (ver Figura 3.6) creados a lo largo de los sitios visitados. El testigo creado más recientemente monitorea al agente móvil, mientras que los más antiguos tienen la responsabilidad de monitorear a los testigos que están más cercanos al agente móvil.

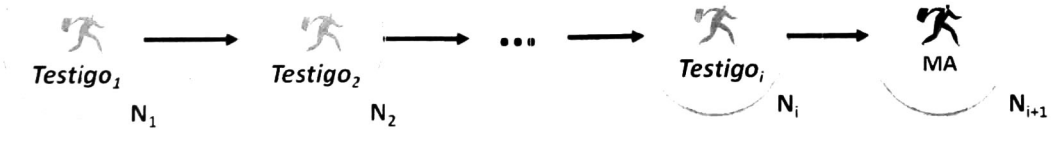


Figura 3.6 Camino de agentes testigo creada por la migración de un agente Móvil (AgMóvil)

Se utilizan varios testigos para que el AgMóvil pueda recuperarse cuando falle en alguno de los sitios visitados (dado que cada sitio tiene una copia del estado del AgMóvil almacenada en el *checkpoint*). Esto permite mantener parte de la ejecución que llevaba antes de fallar ára que no tengan que realizarse todas las tareas desde un inicio. Por lo tanto, esta cadena de testigos es importante. Suponiendo que el agente móvil se encuentra en el sitio N_i, existe un agente $Testigo_{i-1}$ monitoreando al agente móvil y un agente $Testigo_{i-2}$

monitoreando al $Testigo_{i-1}$. Si ocurriera la siguiente secuencia de fallas: el $Testigo_{i-1}$ falla y después falla el agente móvil. Debido a que se pierde el $Testigo_{i-1}$ no existe ningún monitoreo sobre el agente móvil por lo que no podría recuperarse su estado después de una falla. Es por eso que es importante que se recupere también a un agente testigo que haya fallado y mantener la cadena de testigos. El $Testigo_{i-1}$ puede ser recuperado por el $Testigo_{i-2}$ ya que éste se encarga de monitorear su disponibilidad.

La detección de la falla en un agente *testigo* ocurre cuando no se recibe respuesta a un mensaje de monitoreo "*Is_alive?*", como puede observarse en el algoritmo de *Monitoreo()*. Por lo tanto, si se supone que un $Testigo_{i-2}$ falla en recibir respuesta a un mensaje "*Is_alive?*" de un $Testigo_{i-1}$, éste siempre puede suponer que el $Testigo_{i-1}$ ha fallado e iniciar el proceso *recuperar_AgTestigo()* el cual es descrito a continuación para el caso, sin pérdida de generalidad, del $Testigo_{i-2}$.

Algoritmo recuperar_AgTestigo()

1. *Crear agente* p_{i-2}
2. *Hacer*
3. *Enviar SEND(* p_{i-2}, $IDSitio\_Testigo_{i-1}$*)*
4. *Mientras (tiempo_espera_max>0)*
   Si (recibe_MSG("Im_Alive()")) entonces recuperación exitosa
   Fin mientras
5. *Si (tiempo_espera_max==0)*
   Regresar al paso 2
6. *terminar*

Algoritmo de recuperación_ de_ testigo() del agente prueba p_{i-2}

1. *Enviar MSG("Im_Alive()"* $Testigo_{i-2}$*);*
2. *Enviar MSG("Are_U_Alive?"* $Testigo_{i-1}$*),*
3. *Mientras (tiempo_espera_max>0) Esperar (respuesta);*
4. *Si (respuesta_NO_recibida) Crear agente* $Testigo_{i-1}$*;*
5. *Terminar_ejecución();*

De acuerdo al procedimiento *recuperar_AgTestigo()*, una vez que $Testigo_{i-2}$ detecta que un *agente* $Testigo_{i-1}$ ha fallado, se crea a un nuevo agente llamado *agente prueba* p_{i-2}, el cual se utilizará para comprobar que efectivamente el agente testigo que se monitorea ha fallado (y recuperar el agente testigo) y que no se trata de una falsa detección de falla.

Por lo tanto se envía p_{i-2} al sitio donde se encontraba el agente *Testigoi-1,* es decir, se envía a N_{i-1}, para recuperar al agente $Testigo_{i-1}$. El agente p_{i-2} al llegar al sitio N_{i-1} (de acuerdo al algoritmo de *recuperación_de_testigo()*) reenvía un mensaje "*Im_Alive()*" al sitio N_{i-2} para que el agente $Testigo_{i-2}$ continúe en estado de monitoreo normal. Enseguida, verifica si el agente $Testigo_{i-1}$ sigue vivo, si es así, se determina que se trata de una falsa detección de falla y termina con su ejecución. Si el *testigo* no existe, simplemente crea un nuevo Agente $Testigo_{i-1}$ para que monitoree al agente del sitio siguiente (ya sea el agente móvil u otro agente testigo). Así, cuando un agente *testigo* falle, el agente testigo del sitio anterior recuperará su estado.

Sin embargo, en el caso de que fallaran todos los testigos creados y después el Agente Móvil, no habría un testigo disponible para monitorear la falla del agente móvil y por lo tanto recuperar su estado. También si fallara el primer agente *testigo* creado, no habría otro testigo en un sitio anterior que lo monitoree y por lo tanto no podría recuperarse su estado.

3.2.2.3 Disminución en el número de Agentes testigo

Hasta este punto, la estrategia de testigos utilizada es muy similar a la propuesta en (Lyu & Wong, 2003). Sin embargo, cuando el AgMóvil migra a muchos sitios, la probabilidad de que se presenten fallas en múltiples testigos aumenta. El número de mensajes que se intercambian se incrementa con cada testigo creado y además este tipo de dependencia es muy costosa ya que necesita muchos recursos para almacenar los estados intermedios del AgMóvil. Por lo tanto, aún se tienen que resolver 2 problemas:

1. Una cadena de testigos muy larga.
2. Número de mensajes que se intercambian entre los agentes testigo.

♦ **Resolviendo el problema de la cadena de testigos larga**

Se considera que los *testigos* que se han creado más recientemente son más importantes que los testigos creados primero, ya que los *testigos* más antiguos tendrán una copia del AgMóvil en un estado *más antiguo* que los que se han creado en un sitio más cercano al sitio actual (siendo éste el sitio donde se encuentra el AgMóvil y por tanto el estado *más actual* del agente). Para evitar que la cadena de testigos crezca indefinidamente y disminuir la posibilidad de fallas múltiples de testigos, se propone que la cadena de dependencia de los testigos solo crezca hasta un número *n*.

Cabe mencionar que si no se cuenta con el respaldo (*checkpoint*) de los estados intermedios del agente móvil en los sitios visitados, cuando un AgMóvil se pierde debido a la presencia de una falla, éste tendría que iniciar todo su itinerario desde el comienzo. Por lo tanto, se trata de utilizar los respaldos más actuales con el fin de recuperar el último estado consistente de un agente móvil que ha fallado. Si el respaldo más actual no está disponible y se tiene aún un agente testigo que monitorea la falla (y que posee uno de los estados del agente móvil), se realiza la recuperación con el respaldo más reciente disponible y así sucesivamente.

Por lo tanto, *n* representa el número máximo de estados guardados del AgMóvil y desde los cuales se puede recuperar el estado del agente comenzando desde el testigo más reciente. Si se presentan fallas múltiples en los agentes testigo, se podría recuperar hasta el sitio *n* siendo éste el sitio con el testigo más antiguo. Sin embargo, puede presentarse la falla simultánea de todos los testigos creados; en esta situación se tendría el peor de los casos que es iniciar al AgMóvil desde su primera tarea en el itinerario. Elegir un número ***n*** de testigos puede depender de qué tantos estados del agente se desean almacenar, o de la probabilidad que tengan los sitios de fallar al mismo tiempo.

Sin embargo, el estudio para elegir el número óptimo de testigos a utilizar está fuera del alcance de esta tesis; por lo tanto, se supondrá que se trata de un máximo ***n***.

La estrategia empleada para conservar una cadena máxima de ***n*** testigos creados durante la migración del AgMóvil, se describe a continuación:

Estrategia para conservar el número de testigos de un Agente móvil

Paso 1) Cada agente testigo deberá poseer un *atributo* que indique el número de testigo que se refiere (*no_testigo* = *x*), siendo el testigo número 1 el testigo más actual y *n* el testigo más antiguo creado. Cada vez que se crea un nuevo agente testigo, se define este valor como 1 indicando que es el testigo más actual y por lo tanto el más importante ya que posee el último estado consistente del AgMóvil (ver Figura 3.7).

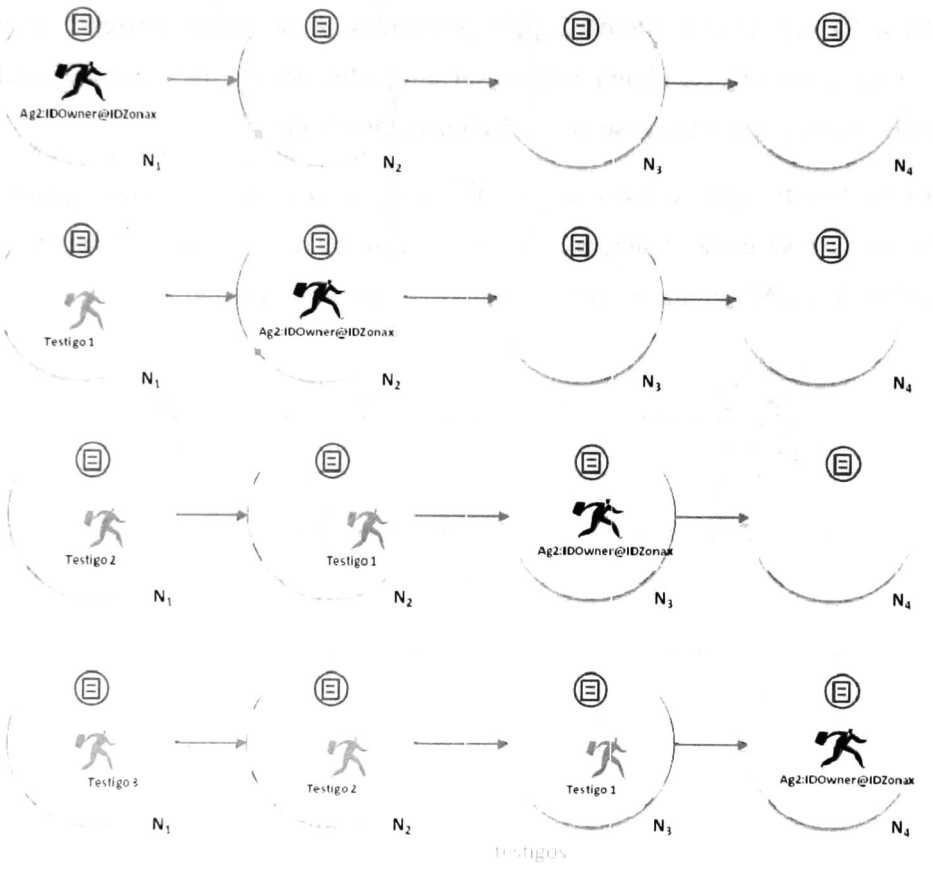


Figura 3.7 Actualización del atributo *no_testigo* para Agentes Testigo$_{no_testigo}$ conforme el AgMóvil migra a nuevos sitios con número máximo de testigos n=3

Paso 2) Para actualizar el número de testigo cuando se crean testigos más actuales durante la migración del AgMóvil, se podría agregar información sobre el número de testigo al mensaje "*Im_Alive*" que se envía durante la etapa de monitoreo. Se propone que si el Agente a quien se está monitoreando recibe un mensaje "*Is_Alive?*",

responda con un mensaje *"Im_Alive(no_testigo)"*, es decir, que se agregue como parámetro el número de testigo que posee.

Paso 3) De ésta forma, cuando un agente recibe un mensaje de un testigo con identificador de testigo (o *no_testigo*) igual o menor que el que posee, éste deberá incrementar el valor de éste atributo, es decir: *no_testigo += 1*.

Paso 4) Cuando el atributo *no_testigo* de un agente testigo es mayor que *n* es decir, que ya se ha llegado el límite de testigos permitidos, dicho agente testigo debe terminar con su ejecución (ver Figura 3.8), no sin antes eliminar el estado guardado del agente móvil debido a que ya no se utilizará en la recuperación.

Con esta estrategia se asegura que solo se tengan a lo más *n* estados guardados del Agente móvil y se evita el desperdicio innecesario de recursos para almacenar estados intermedios del AgMóvil que no se utilizarán para recuperar su estado.

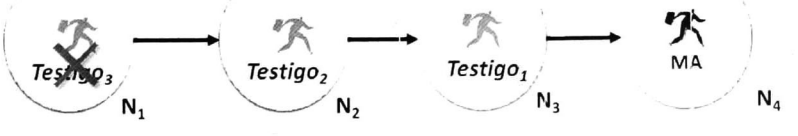


Figura 3.8 Eliminación de un testigo, con una cadena de testigos de n=2

♦ Reduciendo el número de mensajes intercambiados durante el monitoreo

Es claro que al disminuir el número de testigos, el número de mensajes se disminuye. Sin embargo, se propone disminuir un poco más este valor. Debido a que los testigos más actuales son más importantes, la tarea de monitoreo debería hacerse con más frecuencia que aquellos que tienen menos importancia. Por lo tanto, se propone disminuir el número de mensajes intercambiados por los agentes testigo a medida que su *no_testigo* aumenta.

1) Cada agente testigo inicia con tiempo para esperar por el envío de un mensaje de monitoreo (*tmonitor*).

2) El Agente testigo al actualizar su atributo *no_testigo* durante la fase de monitoreo, también deberá duplicar el intervalo de tiempo que espera para el envío de mensajes de

monitoreo. De esta forma, los testigos más actuales monitorean a una frecuencia mayor que los testigos antiguos (ver **¡Error! No se encuentra el origen de la referencia.**).

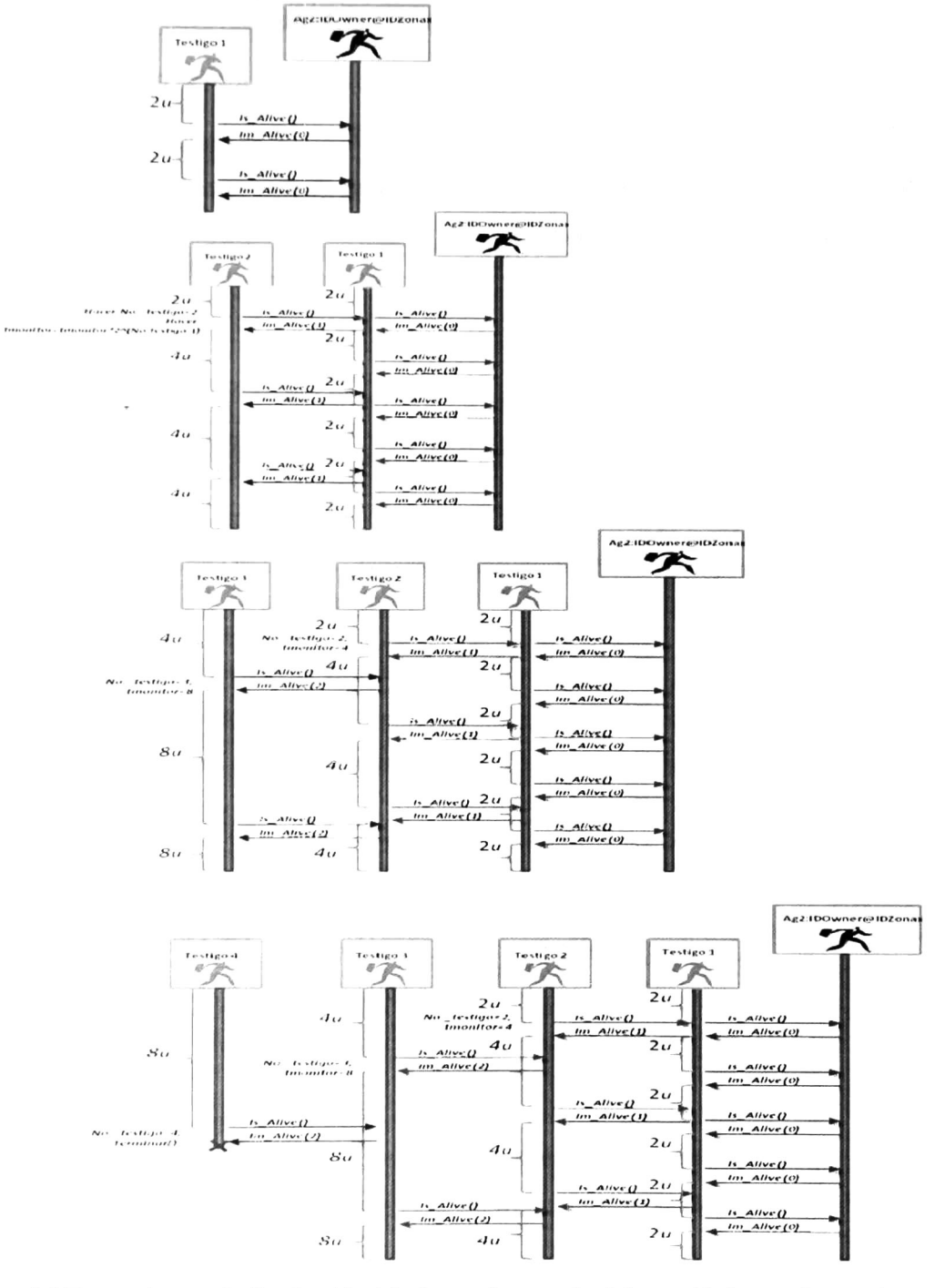


Figura 3.9 Estrategia para la disminución del número de mensajes intercambiados por los agentes testigo, para un número máximo de testigos n=3

Utilizando esta estrategia, se reduce considerablemente el número de mensajes que se intercambian durante el monitoreo, conservando un mayor monitoreo en los testigos que están en sitios "más cerca" al sitio donde encuentra el AgMóvil.

Por lo tanto, el algoritmo de monitoreo de los agentes testigo podría modificarse de la siguiente manera:

Algoritmo de Monitoreo() modificado de cada Agente Testigo para disminuir el número de agentes testigo y el número de mensajes

```
Int no_testigo;
Int tmonitor;
Int tiempo_espera_max;
SITE Sitio_Sig;
ID  Agente; //nombre del agente que se monitorea

hilo Monitoreo() {
    esperar (tmonitor);
    Enviar MSG("Is_Alive?", Sitio_Sig);
    Mientras ( tiempo_espera_max>0 && no_reciba_respuesta())
        Decrementar (tiempo_espera_maximo);
     Fin_mientras;
    Si ( tiempo_espera_max==0) //no se recibió respuesta en un tiempo adecuado
        Si (el agente a monitorear es el AgMóvil)
            detección_falla(); //módulo 3 de detección de fallas
         Si no
            recuperar_AgTestigo();
      Sino //se recibió mensaje Im_Alive(no_testigo1),
        Si(no_testigo1>=no_testigo) //actualizar no de testigo
            no_testigo=no_testigo1+1;
            tmonitor+=tmonitor;
            si(no_testigo>n) //siendo n el número máximo de testigos
              terminar_ejecución();
         fin_si
     fin_si
 }

Función terminar_ejecución(){
```

```
        //eliminar los registros del log del agente que se monitorea
        Eliminar registros del LOG(Agente);
        Eliminar checkpoint(Agente); // eliminar el checkpoing
    End(); //finalizar ejecución
}
Función detección_falla() {
  //a describirse en el módulo3
}
Función recuperar_AgTestigo(){
: //definida anteriormente
}
```

3.2.2.4 Eliminación de Agentes testigo cuando el AgMóvil ha terminado sus tareas dentro de una red.

Una vez que el Agente móvil (AgMóvil) ha terminado su ejecución dentro los sitios de una red, los agentes testigo creados se vuelven innecesarios, por lo tanto deben eliminarse para así liberar los recursos utilizados.

Una manera de eliminar a agentes testigo innecesarios se describió en la sección anterior. Sin embargo, cuando el agente móvil ya no requiere de testigos, la eliminación de los testigos restantes tiene que seguir otra estrategia. Pueden emplearse dos estrategias para eliminar a un agente testigo.

1. *Terminación Activa, por medio de un mensaje de terminación*
2. *Terminación pasiva, mediante la asignación de un tiempo de vida máximo a cada agente testigo*

♦ **Terminación Activa, por medio de un mensaje de terminación**

Para lograr la terminación activa de agentes testigo, cada agente testigo se crea con una función que espera por un mensaje especial de *terminación*(); si este mensaje se recibe, el Agente testigo tendrá que reenviar dicho mensaje de terminación al sitio anterior, propagando así la terminación de los agentes testigo. Finalmente, se eliminan los recursos

utilizados (eliminar los registros del AgMóvil que se monitoreaba, *logs* y *checkpoint*), y se termina su ejecución.

♦ **Terminación pasiva, mediante la asignación de un tiempo de vida máximo a cada agente testigo**

En esta estrategia, a cada agente testigo se le asigna un tiempo máximo de vida (*ttl*) durante su creación, de tal manera que solo permanezca en el sistema máximo ese tiempo. Cuando este tiempo se termine, simplemente libera los recursos utilizados y termina su ejecución. De esta manera, se asegura que el agente testigo no estará ejecutándose de manera infinita y por lo tanto se eliminará eventualmente del sistema.

Algoritmo de Monitoreo() modificado de cada Agente Testigo para disminuir el número de agentes testigo y el número de mensajes

```
Int no_testigo;
Int tmonitor;
Int tiempo_espera_max;
SITE Sitio_Sig;
SITE Sitio_Ant;
ID   Agente; //nombre del agente que se monitorea
hilo Monitoreo() {
    : //descrita anteriormente
  }
Función terminar_ejecución(){
        //eliminar los registros del log del agente que se monitorea
      Eliminar registros del LOG(Agente);
      Eliminar checkpoint(Agente); // eliminar el checkpoing
  End(); //finalizar ejecución
}
Función detección_falla() {
: //a describirse en el módulo 3
}
```

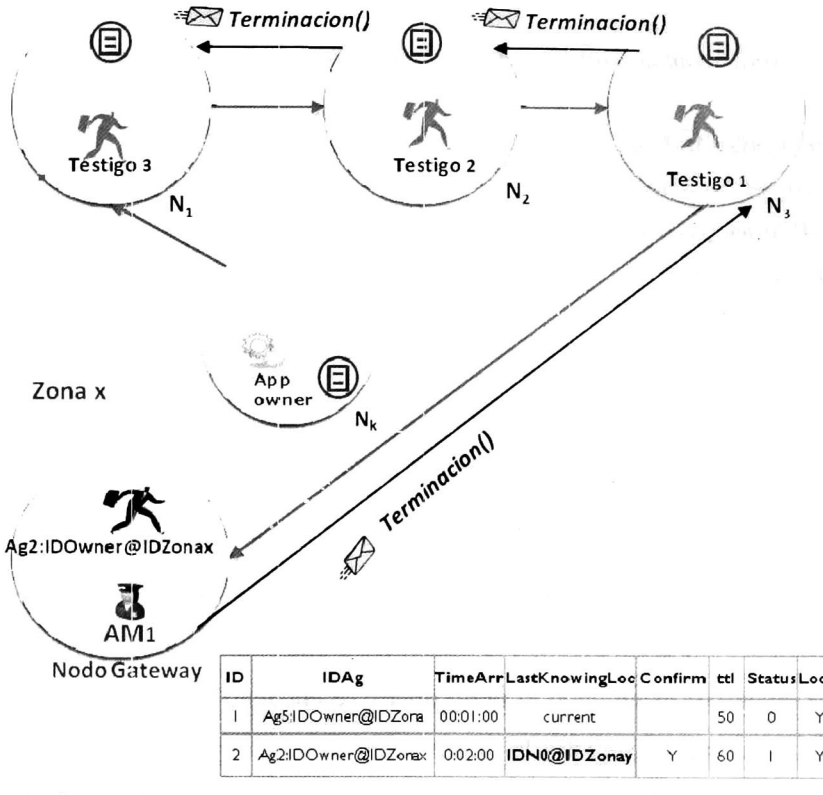
```
Función recuperar_AgTestigo(){
://definida anteriormente
}
Hilo espera_MSG_terminación() //si recibe un mensaje de terminación se elimina de manera activa
{
 Esperar new_msg= MSG();
 Si( new_msg == "terminacion"){
  Enviar MSG(new_msg, Sitio_Ant );
  Terminar_ejecucion();
  }
}
```

3.2.2.5 Monitoreo de Agentes testigo cuando el Agente móvil viaja a un nodo de una zona distinta

Una vez que el AgMóvil ha terminado su ejecución dentro de una zona y requiere migrar a un sitio de una zona distinta, se considera como si el agente móvil ha terminado su ejecución. El AgMóvil es enviado al nodo donde se encuentra el administrador de la zona (AM de la zona) donde se registra la información sobre la zona a la que viajará, según lo descrito en el los protocolos anteriores y se inicia el proceso de terminación de los agentes testigo que se hayan creado en la zona actual.

Como se supone la confiabilidad de los nodos *Gateway*, una vez que el agente móvil ha llegado a este sitio la información en los sitios con agentes testigo se hace innecesaria y por lo tanto tendrían que eliminarse. En este caso, como se describió en la sección anterior, el agente móvil tendría que enviar un mensaje de *"terminación"* al testigo anterior, el cual tendría que reenviarlo a los testigos existentes para que también terminen su ejecución y liberen los recursos reservados con información del agente móvil (ver Figura 3.10).

Cuando el Agente móvil llega al nodo *Gateway* para migrar a una nueva red, se guarda una copia de su estado (*checkpoint*) en almacenamiento estable a la vez que se actualizan las tablas de referencia asociadas a este agente móvil. Por lo tanto, el agente Administrador de la red ahora tomará el rol de agente testigo para este agente móvil. Sin embargo su única responsabilidad será la de asegurar que el agente móvil llegue a su destino.



ID	IDAg	TimeArr	LastKnowingLoc	Confirm	ttl	Status	Local
1	Ag5:IDOwner@IDZona	00:01:00	current		50	0	Y
2	Ag2:IDOwner@IDZonax	0:02:00	IDN0@IDZonay	Y	60	1	Y

Figura 3.10 Monitoreo de agentes testigo cuando el Agente móvil viaja a nodos de la misma zona

Pueden presentarse dos situaciones: 1) que el agente móvil se haya iniciado en un sitio de la zona que maneja el AM de la red (misma zona que su aplicación propietaria), o 2) que el Agente móvil haya iniciado en una zona distinta. Ambas situaciones se describen a continuación.

♦ Comportamiento del AM de la zona actual, cuando el AgMóvil se ha iniciado en la misma zona.

En esta situación, el AM de la zona, es el AM propietario del agente móvil, es decir, el agente responsable de administrar la ejecución del Agente móvil y al cual fue asociado éste al momento de su creación por una aplicación en la red local. Por lo tanto, el AM de la red simplemente guarda una copia del estado del agente móvil y después permite que éste

migre a una nueva red. Sin embargo, como se está migrando a una red externa, espera por un mensaje de confirmación de llegada del agente móvil a la nueva zona, de lo contrario reenviará al Agente móvil hasta recibir confirmación de su llegada.

♦ **Comportamiento del AM de la zona actual, cuando el AgMóvil se inició en una zona distinta**

Cuando se trata de un Agente móvil que fue creado en una zona distinta, el AM de la zona además de guardar copia del estado del agente, debe asegurarse de que éste llegue a su zona destino. Una vez que llega el mensaje de confirmación de que el agente llegó a la siguiente zona, podría enviarse un mensaje al AM de la zona anterior para que libere el espacio ocupado por el *checkpoint* que tiene el estado del agente móvil, ya que éste ya no será necesario para su recuperación.

3.2.3 Detección de fallas

Este módulo se encarga de determinar qué falla es la que ocurrió para así poder manejar la falla de la manera adecuada en el módulo de recuperación de fallas. En las secciones anteriores se describió cómo se va almacenando información para la recuperación de un Agente Móvil (AgMóvil) y qué entidad se encarga de monitorear su ejecución (Figura 3.4). También se describieron estrategias para mantener una cadena de monitoreo recuperando a los agentes testigo que pudieran fallar y terminando a los agentes testigo innecesarios. En esta sección se presentará qué pasa cuando un *agente testigo* no recibe respuesta por parte de un Agente Móvil al mensaje de monitoreo, determinándose así que pudo haber ocurrido una falla en la ejecución del AgMóvil (función *detección_falla() del algoritmo de monitoreo de un agente Testigo*$_i$).

Se supone el siguiente escenario, un AgMóvil en el sitio N_i y un agente *Testigo*$_{i-1}$ en el sitio N_{i-1} que monitorea su ejecución. El agente *Testigo*$_{i-1}$ envía un mensaje "*Is_Alive?*" al AgMóvil pero no recibe respuesta al mensaje de monitoreo.

Pueden presentarse varios casos por las cuales el agente *Testigo*$_{i-1}$ no recibe respuesta de un mensaje "*Is_Alive?*", algunos de estos casos podrían ser:

Caso 1. Que el mensaje de respuesta "*Im_Alive()*" llegue después del tiempo de espera máximo determinado. Se trata de una falsa detección de falla.

Caso 2. Que el mensaje se haya perdido en el camino a su destino debido a un problema en la red.

Caso 3. Que el AgMóvil nunca haya llegado al sitio N_i

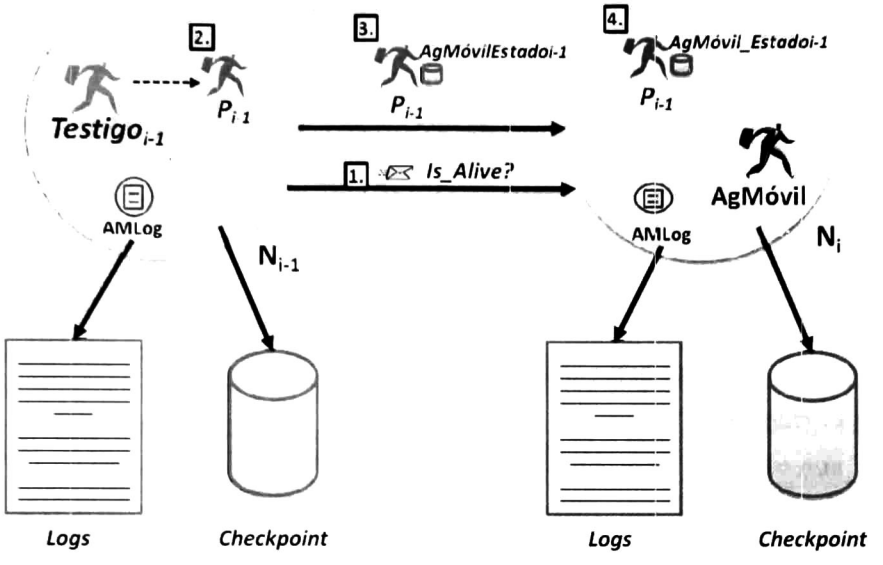
Caso 4. Que el AgMóvil haya llegado correctamente a N_i, pero que haya fallado antes de terminar su ejecución en el nodo.

Caso 5. Que el AgMóvil haya terminado su ejecución en el sitio N_i, pero que haya fallado antes de iniciar a un nuevo agente Testigo.

Caso 6. El AgMóvil terminó su ejecución en el sitio N_i, migró (o no) al sitio N_{i+1}, pero el agente Testigo creado falló justo después de su creación.

Caso 7. El sitio N_i es inaccesible o falló por completo.

Para determinar el tipo de falla de la que se trata, el Agente testigo utiliza a un agente especial llamado *agente prueba* p_{i-1} (el mismo agente que se utiliza para la recuperación de un agente testigo descrito en las secciones anteriores) el cual se envía al sitio donde se debería encontrar el agente móvil; así, después de determinar la falla, el agente p_{i-1} trata de recuperar el estado del AgMóvil. Por lo tanto, como es posible que el agente p_{i-1} recupere a un AgMóvil que ha fallado, debe de viajar al sitio siguiente (N_i) con la copia del último estado almacenado (*checkpoint*) del AgMóvil, el cual se encuentra en almacenamiento estable en el sitio N_{i-1}. En la Figura 3.11 se describe ésta situación donde, el $Testigo_{i-1}$ envía un mensaje de monitoreo al AgMóvil (1.), pero al no recibir respuesta, inicia a un agente p_{i-1} (2.) el cual viaja (junto con la copia del estado anterior del AgMóvil en el *checkpoint* 3.) al sitio donde debería estar el AgMóvil para así poder determinar la falla y recuperar al AgMóvil en caso de que se haya perdido su estado (4.).



1. *El Agente Testigoi-1 envía un mensaje Is_Alive? al AgMóvil en el sitio Ni*
2. *Como no se recibió respuesta, el Testigoi-1 crea a un agente pi-1 para determinar la falla y recuperar al AgMóvil*
3. *El agente pi-1 viaja al sitio Ni junto con una copia del último estado guardado del AgMóvil (checkpoint)*
4. *El agente pi-1 llega al sitio Ni para determinar la falla*

Figura 3.11 Creación de un Agente p_{i-1} para determinar el tipo de falla en la ejecución de un AgMóvil

Por lo tanto, en este módulo la función del Agente *Testigo$_{i-1}$* se describe mediante el siguiente algoritmo:

Algoritmo detección_falla() de un agente Testigo$_{i-1}$

```
//atributos
Int no_testigo;
Int tmonitor;
Int tiempo_espera_max;
SITE Sitio_Sig;
SITE Sitio_Ant;
ID   Agente; //nombre del agente que se monitorea

Función detección_falla() {
  crear Agente pi-1
```

```
fijar no_intentos_max;
hacer {
    Enviar SEND(pi-1, Sitio_Sig); //enviar el agente prueba al sitio del AgMóvil
    Mientras (tiempo_espera_max>0) {
      Si(recibe_MSG("Im_Alive()")) entonces recuperación exitosa;
    }
   Si(tiempo_espera_max==0 && no_intentos_max>0) //no se recibió mensaje
       Decrementar no_intentos_max;
} mientras (no_intentos_max>0);
Si (no_intentos_max==0 && no_recibio_MSG("Im Alive()")) // el sitio es inaccesible
    Recuperar_AgMóvil_sitio_Innaccesible(); // caso 7, a describirse en el modulo 4
}
```

Una vez en el sitio N_i, el agente prueba realiza el siguiente algoritmo:

Algoritmo determinación_de_falla() de un agente p_{i-1} al llegar al sitio N_i

```
Función determinación_de_falla(){
      // mensaje para que el ag Testigoi-1 sepa que el sitio siguiente es accesible.
  Enviar MSG("Im_Alive()",Testigo_{i-1});
  Enviar MSG("Are_U_Alive",AgMóvil); //determinar si el AgMóvil sigue vivo
  Mientras (tiempo_espera_max>0) esperar(respuesta);
 Si(respuesta=="Im_Alive()") //el AgMóvil sigue en ejecución, es una falsa detección
      Terminar_ejecución(); // caso 1 y 2
 Sino{    // el AgMóvil no está en ejecución por lo tanto ha fallado y requiere recuperación
   Si ( buscar( LOG, "entrada", AgMóvil) NO existe){
       //el AgMóvil NO se registro en el LOG por lo tanto se perdió antes de llegar al sitio Ni
       Recuperar_AgMóvil_CheckpointAnterior(checkpoint); // caso 3, en el modulo 4
     }
   Sino_ Si (buscar(LOG, "salida", AgMóvil) existe){
      // hay registro de salida, por lo tanto o el AgMóvil fallo antes de migrar al nodo Ni+1 y
      // NO creó al Testigoi ó  El AgMóvil migro al sitio siguiente y el testigo fallo
       Enviar MSG("Are_U_Alive",Testigoi); //determinar si el Testigoi está vivo
       Mientras (tiempo_espera_max>0) esperar(respuesta);
       Si(respuesta=="Im_Alive()") //el Testigoi está en ejecución, es una falsa detección
```

```
        Terminar_ejecución(); //caso 1 y 2
      Sino {
        // no hay Testigoi pero si hay registro de salida, por tanto el AgMóvil pudo fallar
        // antes de migrar al N(i+1), o el que falló fue el Testigoi, se debe recuperar Testigo
        Crear agente Testigo(i-1)(no_testigo=1, sitio_monitorear=Ni+1,
                                  Agente_monitorear= AgMóvil); // caso 5 y 6
          }
      }
    Sino { //no hay registro de salida, el AgMóvil falló antes de terminar su ejecución
        Recuperar_AgMóvil_CheckpointAnterioryLogs(checkpoint,LOG); //caso 4
        }
  }
 Terminar_ejecución();
}//fin funcion
```

De acuerdo a los algoritmos anteriores se tiene que, para cada uno de los casos:

- **Caso 1. El mensaje de respuesta "Im_Alive" llegó después de un tiempo de espera máximo.**

  Cada agente *Testigo* espera un tiempo máximo por la respuesta a un mensaje "*Is_Alive?*", pero si dicho tiempo ha expirado, el agente $Testigo_{i-1}$ inicia el proceso *detección_falla()*, en el cual se crea a un Agente prueba p_{i-1} y se envía al sitio N_i. Una vez en el sitio, el agente p_{i-1} envía un mensaje "Are_U_Alive?"al AgMóvil, el cual le responde con un mensaje "Im_Alive()", por lo que el agente prueba p_{i-1} determina que se trata de una falsa detección de falla y termina su ejecución.

- **Caso 2. El mensaje se perdió debido a un enlace no confiable en la red o, el mensaje llegó al sitio N_{i-1} después de un tiempo de espera máximo.**

  Se realizan las mismas operaciones que para el caso 1, porque ya sea que el mensaje haya llegado después de un tiempo determinado o que se haya perdido, el $Testigo_{i-1}$ envía a un agente prueba que determina que el AgMóvil sigue en ejecución por lo que no ha ocurrido una falla. Aunque, para evitar que se vuelva a mandar a un agente

prueba debido a que los mensajes sufren un retraso alto, podría incrementarse el tiempo de espera por el mensaje *"Im_Alive()"*

- **Caso 3. El agente móvil nunca llegó al sitio N_i. Puede ser que el AgMóvil falló cuando estaba listo para migrar al sitio N_i o que haya fallado durante la migración.**

  El agente $Testigo_{i-1}$ crea a un agente prueba p_{i-1} y la envía al sitio N_i. El agente p_{i-1} al llegar al sitio N_i verifica si el Agente móvil se encuentra en ejecución en ese sitio, al no encontrarlo verifica la tabla de arribo de los agentes (en el LOG) y determina que no hay registro de la entrada del agente, por lo tanto determina que **el agente móvil debió perderse en el camino a N_i.** En este punto, la información guardada en el *checkpoint* en el sitio N_{i-1} puede utilizarse para recuperar el estado del agente móvil, por lo tanto, el agente prueba p_{i-1} deberá viajar con los datos necesarios para recuperar el estado del agente móvil. En este caso, el agente prueba p_{i-1} inicia la función ***Recuperar_AgMóvil_CheckpointAnterior(checkpoint)***, en esta función (que se describirá en la siguiente sección)

- **Caso 4. El agente Móvil llegó correctamente a N_i pero falló antes de terminar su ejecución en el sitio. Por lo tanto el AgMóvil falló justo después de registrar su entrada en el sitio N_i o falló antes de registrar su salida.**

  El agente $Testigo_{i-1}$ crea a un agente prueba p_{i-1} y la envía al sitio N_i. El agente p_{i-1} al llegar al sitio N_i verifica si el Agente móvil se encuentra en ejecución en ese sitio, al no encontrarlo verifica la tabla de arribo de los agentes (LOG) y determina que sí hay *registro de su entrada.* Ahora, el agente p_{i-1} revisa si hay un *registro de la salida* del agente móvil y al no encontrarlo determina que ha fallado durante su ejecución N_i.

  En este caso el agente prueba p_{i-1} inicia la función **Recuperar_AgMóvil_CheckpointAnterioryLogs(checkpoint,LOG)**, descrita a detalle en la sección siguiente. Esta función recuperará al AgMóvil con el estado guardado en el *checkpoint* del sitio anterior, después actualizará el estado del AgMóvil

hasta la última operación registrada en el archivo LOG del sitio actual (N_i). Para finalmente dejar que el AgMóvil continúe con su ejecución normal.

- **Caso 5. El agente móvil falla justo después de registrar su salida del sitio N_i. No se creó un nuevo agente testigo.**

  Se ha enviado un agente prueba p_{i-1} al sitio N_i. El agente p_{i-1} verifica si el Agente móvil se encuentra en ejecución en ese sitio, al no encontrarlo verifica la tabla de arribo de los agentes y determina que sí hay un *registro de entrada*. Ahora, el agente p_{i-1} revisa la tabla de registros y determina que sí hay un *registro de la salida* del agente móvil.

  En esta situación pueden haberse presentado dos tipos de fallas: 1) que el agente móvil haya fallado justo antes de crear al Agente Testigo y antes de migrar al sitio siguiente, o 2) que el agente testigo creado haya fallado justo después de su creación.

  Ambos casos se resuelven con solo **iniciar al Agente *Testigo*$_i$**, ya que al recuperar al agente testigo, éste se encargará de monitorear al AgMóvil en el sitio siguiente. Si el AgMóvil falló durante su migración o antes de migrar, el Testigo$_i$ detectará la falla y tratará de recuperar al AgMóvil en el sitio siguiente. Si el AgMóvil no falló y lo que falló fue el Testigo$_i$, cuando se recupera el Agente testigo éste continúa con el monitoreo normal del Agente Móvil.

- **Caso 6. El agente Móvil terminó su ejecución en N_i y migró (o falló antes de migrar) al sitio N_{i+1} y el agente testigo en el sitio N_i falló justo después de ser creado.**

  El escenario de falla para este caso se describió en el caso 5.

- **Caso 7. El sitio N_i es inaccesible o ha fallado por completo**

  El Agente Testigo$_{i-1}$ crea a un Agente prueba p_{i-1} y se envía al sitio N_i. El agente Testigo$_{i-1}$ espera un *tiempo máximo* por la llegada de un mensaje de confirmación "Im_Alive()" pero el mensaje no es recibido. Aún puede pasar que el agente p_{i-1} haya fallado durante su migración a N_i, por lo tanto el agente prueba se envía nuevamente al sitio N_i durante un número determinado de intentos. Después de agotar todos los intentos de mandar al agente p_{i-1} a N_i, el agente Testigo$_{i-1}$ finalmente determina que el

sitio donde se ejecutaba el AgMóvil ha fallado o se vuelto inaccesible por lo es necesario recuperar al AgMóvil en el sitio N_{i-1} para que busque un sitio alternativo para migrar (función **Recuperar_AgMóvil_sitio_Innaccesible()** a describirse en la siguiente sección).

En esta sección se han descrito los escenarios de falla de un AgMóvil detectados por un Agente Testigo en el sitio anterior. En la siguiente sección se detallarán las funciones de Recuperación de un AgMóvil.

3.2.4. Recuperación de fallas

De acuerdo a la sección anterior, se tiene que el AgMóvil falló en el sitio N_i, y que la falla fue detectada por el agente $Testigo_{i-1}$ en el sitio anterior N_{i-1}. Ahora se definirán las tres funciones de recuperación de los casos detectados.

- **Caso 3. Falla: El agente móvil nunca llegó al sitio N_i. Puede ser que el AgMóvil falló cuando estaba listo para migrar al sitio N_i o que haya fallado durante la migración.**

Cuando una falla de este tipo es detectada, solo basta con iniciar al AgMóvil que falló en el sitio donde debería de encontrarse N_i. Como su último estado guardado viajó con el agente prueba p_{i-1} a N_i, el agente prueba solo tiene que crear nuevamente al AgMóvil utilizando la información del *checkpoint* que tomó del sitio anterior N_{i-1}. El algoritmo que describe la recuperación de esta falla se describe a continuación.

---

*Función **Recuperar_AgMóvil_CheckpointAnterior(checkpoint)**{*

//de un agente prueba p_{i-1} en N_{i-1}

Crear Agente (AgMóvil,checkpoint); //crear al AgMóvil desde el checkpoint

Iniciar Agente AgMóvil.iniciar(); //se inicia el AgMóvil, la primera función que debería ejecutar es

// registrar su entrada en el LOG

Terminar_ejecucion();

}

- **Caso 4. Falla: El agente Móvil llegó correctamente a N_i pero falló antes de terminar su ejecución en el sitio. Por lo tanto el AgMóvil falló después de registrar su entrada en el sitio N_i.**

Para que el agente prueba p_{i-1} recupere al AgMóvil a partir de que se determina esta falla, necesita:

1. La información del *checkpoint* tomado del sitio anterior para primero recuperar el estado del AgMóvil al momento de entrar al sitio N_i.
2. Una vez que se tiene el estado del AgMóvil al momento de entrar a N_i, se requiere actualizar el estado del agente hasta el momento antes de presentarse la falla. Para ello se puede utilizar la información almacenada en los registros LOG del sitio N_i (comenzando desde la primera operación que está justo después del LOG para el registro de entrada del AgMóvil al sitio N_i), en el cual se registraron las operaciones que el AgMóvil fue realizando.
3. Después de tener el estado actualizado del AgMóvil, simplemente hay que reiniciar su ejecución.

La función de recuperación del AgMóvil al presentarse este tipo de falla puede observarse en el siguiente algoritmo.

Función ***Recuperar_AgMóvil_CheckpointAnterioryLogs(checkpoint, LOG)*** *{*

//de un agente prueba p_{i-1} en N_{i-1}

Crear Agente (AgMóvil, checkpoint);

/ Rollback(LOG, AgMóvil); */*

Actualizar_estado_replay(AgMóvil, LOG); //actualizar el estado del AgMóvil usando el LOG

Iniciar Agente AgMóvil.iniciar();

Terminar_ejecucion();

}

Otra opción que puede realizarse, en lugar de actualizar el estado del AgMóvil usando los LOGs del sitio N_i, es la de utilizar la información de los LOGs para deshacer las operaciones que el AgMóvil realizó hasta antes de registrar su entrada al sitio N_i. Para esto

se supondría la existencia de una función llamada ***Rollback(LOG,AgMóvil)*** la cual iría tomando registro por registro del LOG (comenzando desde el último registro almacenado por el AgMóvil) y realizaría una operación tal que dejaría al sitio N_i sin los efectos causados por la operación del AgMóvil. Sin embargo, debería suponerse también, que estas operaciones siempre pueden deshacerse.

- **Caso 7. El sitio N_i es inaccesible o ha fallado por completo**

Este tipo de falla es la más indeseable, ya que es difícil determinar si el AgMóvil realmente ha terminado con su ejecución o si el sitio donde se ejecuta se ha vuelto temporalmente inaccesible. Si solo se trata de un sitio temporalmente inaccesible, significa que si se crea una nueva instancia del AgMóvil se estaría violando la propiedad *exactamente_una_ejecución* de los agentes, ya que se estarían ejecutando dos AgMóvil al mismo tiempo en sitios distintos. Por lo tanto, lo más importante es asegurarse de que el AgMóvil ha fallado completamente lo cual no siempre es fácil de determinar con los protocolos propuestos por otros autores.

Sin embargo, si se utiliza la información generada en los protocolos descritos en el capítulo 2, se podría tener un tiempo máximo para determinar si el AgMóvil ha fallado realmente. Para lograrlo simplemente hay que esperar a que el tiempo de vida (*ttl*) del AgMóvil se agote (utilizado en el protocolo de control de agentes).

Una vez que transcurre el *ttl* del AgMóvil y éste no se ha comunicado con el Agente Administrador (AM) de la red para renovar su tiempo de vida. Podría determinarse que el AgMóvil efectivamente ha fallado y en esa situación iniciar un nuevo AgMóvil' utilizando el último *checkpoint* almacenado.

Por lo tanto se podría realizar la siguiente función para recuperar al AgMóvil.

SITE Sitio_Sig;
SITE Sitio_Ant;
Int tiempo_max;
Función ***Recuperar_AgMóvil_sitio_Inaccesible()*** *{ //de un agente* $Testigop_{i-1}$ *en* N_{i-1}
Enviar MSG("Site_ Unreachable(Sitio_Sig, AgMóvil, IDCheckpoint)",AM);
While(tiempo_max>0) esperar(respuesta);

```
Si (respuesta == "Recover_AgMóvil"){
   Crear Agente (AgMóvil', respuesta.getTtl(), respuesta.getNew_Name(), checkpoint);
   Iniciar agente AgMóvil'.iniciar();
  }
Si (respuesta== "Terminate"){
   terminar_ejecución();
 }}
```

Aún si el AgMóvil original quisiera renovar su ttl mediante un mensaje de *New_ttl_Req* al AM de la red, éste ya no lo renovaría por lo que el AgMóvil original terminaría su ejecución.

En este capítulo se han descrito las partes principales del protocolo de recuperación de agentes móviles, sin embargo, aún no se ha considerado el caso de que se tuvieran sitios que fallaran por un periodo de tiempo y luego se recuperaran. Para este caso, se tendrían que definir estrategias para eliminar la información que se guardó de los agentes móviles que estaban ejecutándose en ese sitio y que ya no son necesarias.

También podría presentarse el caso de que el AgMóvil y todos los agentes testigo creados fallaran simultáneamente. En esta situación, con el protocolo de control de agentes, se podría detectar que el AgMóvil ha fallado en un máximo de *ttl* unidades de tiempo (lo cual no puede lograrse con los protocolos de recuperación de agentes descritos por otros autores), ya que su tiempo de vida *ttl* se ha agotado y no se ha comunicado con el AM de la zona. En este momento el AM de la zona iniciaría el proceso de localización del agente móvil y podría, en un momento dado, iniciar el proceso de recuperación del Agente móvil en el último sitio donde se encontrara un *checkpoint* de éste o de lo contrario, utilizar el *checkpoint* guardado cuando el agente entró por primera vez a la zona, e iniciar el AgMóvil desde ese punto. Este caso no es deseado ya que se estarían ejecutando varias veces las operaciones realizadas por el AgMóvil, lo cual violaría la propiedad de *exactamentee-una-ejecución*; por lo tanto tendría que definirse una función ***Rollback()*** que deshiciera las operaciones que realizo el AgMóvil en cada sitio bajo la suposición de que todas las operaciones que el AgMóvil realiza siempre pueden deshacerse.

Capítulo 4.

Modelado de Protocolos de Control de Agentes

Resumen. En este capítulo se especifican los protocolos propuestos mediante una extensión de redes de Petri temporizadas, llamada *Redes de Petri con Cronómetros Globales*. Se presenta la definición del formalismo y se describen los modelos de los protocolos usando el formalismo propuesto.

4.1 StopWatch Petri Nets

4.1.1 Generalidades

Las Redes de Petri Temporizadas (**TPN**- Timed Petri Nets) proporcionan un ambiente uniforme para modelado, diseño y análisis de desempeño de los sistemas de eventos discretos. Nos ayudan a modelar tareas u operaciones incluyendo el *tiempo* o duración de las mismas en los modelos (Wang, 1998). Sin embargo, las TPN no son capaces de modelar Sistemas Interrumpibles (Allahham & Alla, 2007); dichos sistemas se encuentran a menudo en los Sistemas de Tiempo Real y se componen de varias tareas que interactúan entre sí cuyo comportamiento puede ser interrumpido/restablecido en cierto tiempo.

Por ejemplo, se considera un sistema donde una pieza va en una banda transportadora de un punto *A* a un punto *B* en 10 unidades de tiempo y nos interesa considerar, en el modelo, si la pieza puede llegar al punto *B* en un intervalo de tiempo dado (por ejemplo [10,12]) para considerar el tiempo que la banda pudiera permanecer detenida debido a fallas impredecibles durante el transporte (ver Figura 4.1.a).

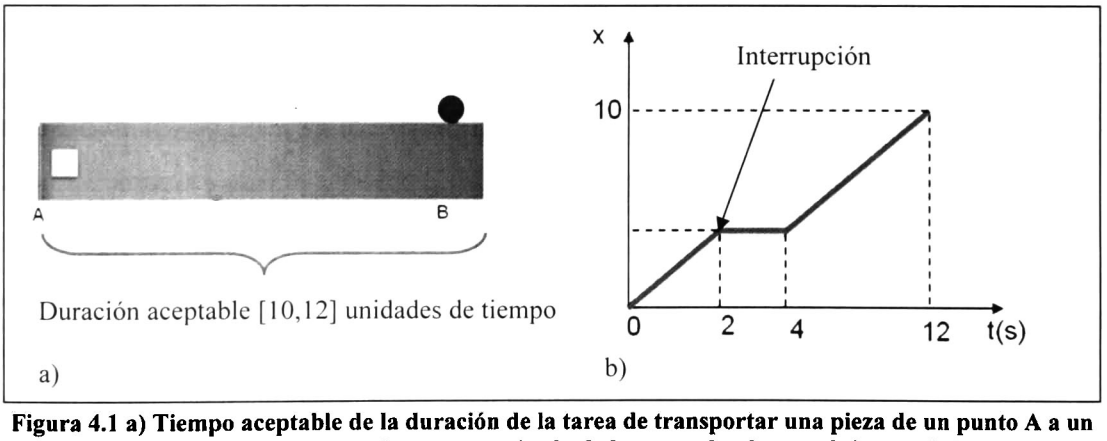


Figura 4.1 a) Tiempo aceptable de la duración de la tarea de transportar una pieza de un punto A a un punto B, b) Gráfica del comportamiento temporizado de la tarea, donde *x* es el tiempo de la tarea y t(s) es el tiempo normal.

Ahora, si se deseara modelar este problema utilizando TPN, el modelo obtenido podría ser el representado en la Figura 4.2 a donde la tarea en el lugar *P2* se interrumpe al disparar *t3*, pero, como se observa, el restablecimiento de la tarea debería darse al disparar *t4* sin embargo el estado alcanzado antes de la interrupción se pierde ya que *t2* se habilitará nuevamente desde 0 y no desde el valor previo alcanzado (ver Figura 4.2 b), de acuerdo a las reglas de habilitación y disparo de las TPN (Wang, 1998).

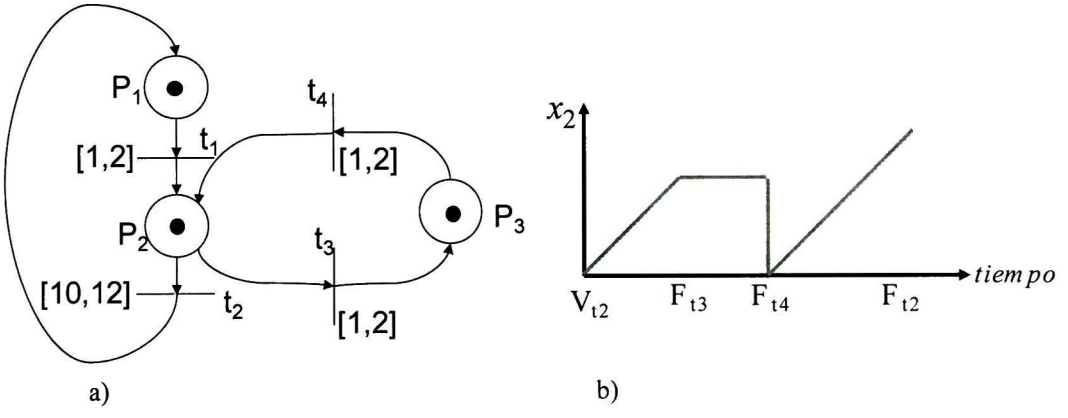


Figura 4.2 a) TPN del problema 1, b) Gráfica de variación de tiempo de t2, donde x2 es el reloj asociado al lugar P2, F_{ti} indica el disparo de una transición ti.

Es de gran utilidad el modelar el comportamiento *interrumpible* de las tareas y las TPN no son adecuadas para capturar dicho comportamiento; para hacerlo se requiere el uso de las Redes de Petri (PN) con relojes que puedan ser *suspendidos* y luego *restablecidos* al valor que tenían antes de ser interrumpidos; además con la capacidad de ser reinicializados a un valor dado (por ejemplo, cero).

Las Stop Watch Petri Nets (SWPN) como su nombre lo indica, utilizan cronómetros que pueden ser detenidos y restablecidos. Se presentan como una extensión a las TPN para abordar el modelado de tareas o acciones que pueden ser suspendidas, por un periodo de tiempo, y restablecidas después justo en el estado en el que estaban antes de ser suspendidas.

Varios autores han propuesto extensiones a las TPN para obtener SWPN soportando el modelado de tareas interrumpibles. En (Magnin et al., 2006a; Magnin et al., 2006b) se

propone una extensión a las TPN utilizando arcos inhibidores y arcos de flujo. Los arcos inhibidores les permiten modelar prioridades entre las tareas (las cuales se modelan mediante transiciones temporizadas), donde el cronómetro asociado a una transición es detenido cuando la transición se encuentra inhibida, y restablecida cuando la inhibición desaparece. Se asocia un cronómetro por cada transición y éste representa el tiempo transcurrido durante el cual la transición asociada fue habilitada por última vez (por el marcado) y durante el cual la transición no está inhibida. Un ejemplo de este tipo de redes es el mostrado en la Figura 4.3, donde primero, los arcos inhibidores prohíben el disparo de t3 y, de acuerdo a la temporización de las transiciones, se permite el disparo de t2 antes que t1. Una vez que se dispara t2, la tarea en t3 se restablece.

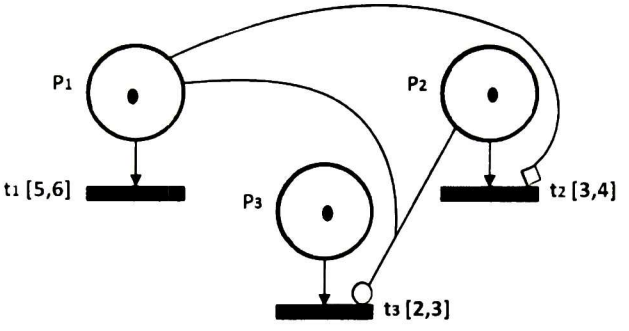


Figura 4.3 Ejemplo de una SWPN con arcos de flujo y arcos inhibidores

En (Lime & Roux, 2003; Magnin et al., 2005;Magnin et al, 2006c) se define la semántica para una extensión a las TPN para planificadores (*Scheduling-TPN*), la cual consiste en enriquecer al modelo de las TPNs con políticas de planificación (*scheduling*), es decir, la manera en que los diferentes planificadores del sistema activan o suspenden tareas. Además, se asocia a cada lugar información sobre el procesador donde se realizará la tarea y la prioridad de la misma; así, si dos tareas se realizan en el mismo procesador en un mismo tiempo y una de ellas tiene prioridad sobre la otra, la tarea con menor prioridad se suspende, modelando así el comportamiento interrumpible, para posteriormente reanudarla cuando la tarea con mayor prioridad haya finalizado. Por ejemplo, en la Figura 4.4 se muestra un ejemplo de una *Scheduling-TPN* en la cual, de acuerdo al marcado inicial, las tareas en los lugares P1 y P3 se realizan en el mismo procesador ( ), pero, debido a que la tarea en P3 tiene prioridad mayor, la primera transición a dispararse será t3.

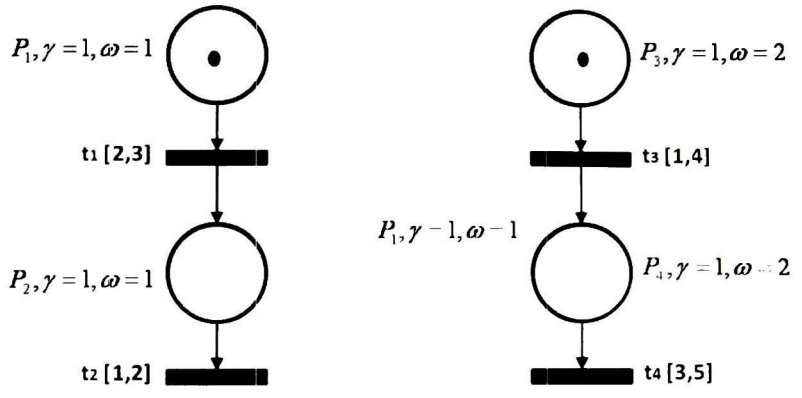


Figura 4.4 Scheduling-TPN para dos tareas en un procesador

En la extensión propuesta por (Allahham & Alla, 2007), se sigue un enfoque llamado *Post and Pre-initialized Stopwatch Petri Nets*, es decir, SWPN post y pre-inicializadas. El concepto básicamente es el mismo que otros autores, es asociado un cronómetro (o stopwatch) a una sola transición, pero introduce un mecanismo para reinicializar los cronómetros. El cronómetro asociado a una transición (la cual se dibuja con un trazo más grueso), es suspendido cuando el lugar de entrada a dicha transición es desmarcado, y restaurado cuando la marca regresa al lugar. Por ejemplo, en la Figura 4.5 se muestra un ejemplo de una SWPN propuesta por Allahham, en la cual, la transición t2 es la única que tiene asociado un cronómetro, y el resto de las transiciones poseen una temporización normal a las TPN. P1 representa la ejecución de la tarea, mientras que P2 representa el estado donde la tarea está interrumpida. Las transiciones t3 y t4 representan la ocurrencia de la interrupción y la reanudación de la tarea respectivamente. A la transición t2 le es asociado el cronómetro x_2, cuando esta transición es disparada dicho cronómetro es reinicializado a cero.

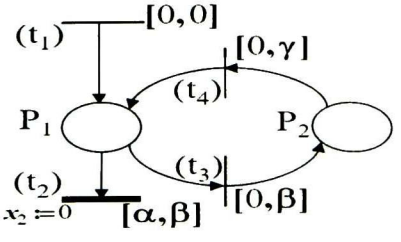


Figura 4.5 Ejemplo de una *post and pre-initialized* SWPN de una tarea interrumpible

Una definición similar a las redes de Allahham son propuestas en (Berthomieu et al., 2007), donde cada transición asociada a cronómetro posee un tipo especial de arco, el cual, en lugar de tener punta de flecha posee punta de cuadro. En este tipo de redes, de manera similar a Allahham, cuando el lugar de entrada a una transición con un cronómetro pierde la marca que habilitaba la transición, el cronómetro se detiene, y reanuda su funcionamiento cuando el lugar es marcado de nuevo (ver Figura 4.6). La diferencia con la definición de Allahham es que no se utiliza el mecanismo para reinicializar los cronómetros.

Todas las extensiones mencionadas tienen la particularidad de que sólo se utiliza un cronómetro (asociado a una transición) para modelar la ejecución de una tarea, pero si se desearan modelar las diferentes partes, fases o estados que componen esa tarea (la cual podría ser interrumpida en cualquier momento, independientemente de la fase en la que se encuentre), no sería posible. Se tendría que modelar cada fase como una tarea con un cronómetro y luego otra fase con otro cronómetro y así sucesivamente.

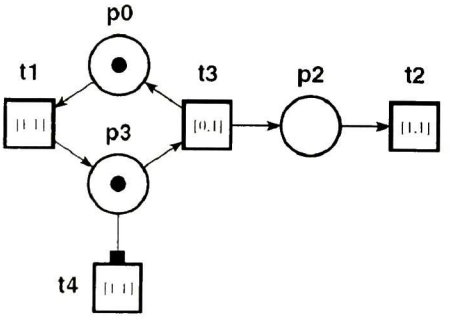


Figura 4.6 Ejemplo de una SWPN donde la transición que tiene un cronómetro es asociada a un arco con punta de cuadro

Dadas las limitaciones que presentan los formalismos anteriores, en esta tesis se propone una extensión a las TPN basada en las SWPN definidas en (Allahham & Alla, 2007) la cual incluye una semántica similar a la utilizada por Cassez en (Cassez & Roux, 2006) para el modelado de Autómatas con Cronómetros. Esta extensión nos permite el uso de varios cronómetros (StopWatches) manipulados desde varios lugares (cronómetros globales) de manera que una tarea pueda ser modelada en sus diferentes estados por un solo cronómetro, el cual modela el tiempo de ejecución de dicha tarea. Además se incorpora el uso de restricciones sobre los cronómetros asociados a las transiciones, modificando así las

reglas de habilitación de las mismas. Los modelos que pueden construirse con la extensión propuesta capturan comportamientos más complejos, por lo que las definiciones previas (Allahham & Alla, 2007; Berthomieu et al., 2007) resultan subclases de este nuevo formalismo, el cual se le llamará *Redes de Petri con Cronómetros Globales (GSWPN)*.

4.1.2 Definición Formal de las GSWPN

Definición 1. Una PN con cronómetros globales es un tupla (P, T, Pre, Post, Mo, X, ϕ, G, INI) donde:

- P= {p1,p2, … , pn} y T = {t1,t2, … tm} son los conjuntos finitos de lugares y transiciones de la PN, donde $P \cap T = \emptyset; P \cup T \neq \emptyset$
- Pre : $P \times T \rightarrow \{0,1\}$ es una función de incidencia previa la cual representa los arcos de entrada a las transiciones, siendo 0 cuando el arco no está definido.
    - $\bullet t_i$ es el conjunto de lugares que están a la entrada de t_i.
    - $\bullet p_i$ es el conjunto de transiciones que están a la entrada de p_i.
- Post : $P \times T \rightarrow \{0,1\}$ es una función de incidencia posterior que representa los arcos de salida de las transiciones.
    - t_i es conjunto de lugares que están a la salida de t_i.
    - $p_i \bullet$ es conjunto de transiciones que están a la salida de p_i.
- Mo$\in\{0,1\}^{|P|}$ es el marcado inicial de la red, asociando a cada lugar p, siendo $M(p_i) = 1$ si el lugar está marcado, o $M(p_i) = 0$ si el lugar p_i no tiene marcas.
- $X = \{x_i | i \in \mathbb{N}\}$ es el conjunto finito de cronómetros, el cual tiene los siguiente atributos:
    - $\dot{X} = \{(\dot{x}_i = c) | c \in \{0,1\}\}$ es el conjunto de las derivadas de los cronómetros con respecto al tiempo
    - $C(X) = \{(x_i \; op \; v_j) \mid op \in \{=, <, >, \leq, \geq\} \; and \; v_j \in \mathbb{R}^{\geq 0}\}$ es el conjunto de restricciones sobre los cronómetros en X.

- $\phi : P \rightarrow 2^{\dot{X} \cup C(X)}$ es la función que asocia a los lugares un subconjunto de derivadas de los cronómetros y de restricciones sobre los mismos (invariantes); de esta forma $\phi(p_i) = \{(\dot{x}_1 = c_1), \ldots, (\dot{x}_k = c_k), (x_1\ op_1\ v_1), \ldots, (x_j\ op_j\ v_j)\}$ donde $p_i \in P$.
- $G: T \rightarrow Exp_G(C(X))$ es una función que asocia a cada transición de la red una expresión booleana (guarda) definiendo las condiciones que deben cumplirse para disparar dicha transición.
- $INI : T \rightarrow Exp_{INI}(X)$ es una función que asocia a cada una de las transiciones de la PN un conjunto de expresiones de la forma $(x_i := v_j)$, estas expresiones representan el valor que se le asigna al cronómetro cuando la transición a la que se asocian es disparada, es decir, con estas expresiones se pueden reinicializar un cronómetro a un valor deseado.

En este tipo de redes cada cronómetro x_i puede ser interrumpido o suspendido $(\dot{x}_i = 0)$ y reactivado después $(\dot{x}_i = 1)$ justo en el mismo valor que tenía antes de ser interrumpido. Esto se declara para cada lugar de la red. La función ϕ asocia a cada lugar: 1) un subconjunto de derivadas para los cronómetros respectivos en el lugar, el cual indica si los cronómetros asociados están activos o interrumpidos cuando el lugar está marcado; y 2) un conjunto de restricciones sobre los cronómetros asociados, las cuales representan el invariante en el tiempo donde la transición puede ser habilitada, en otras palabras, indican cuándo la marca está disponible para poder habilitar las transiciones de salida a dicho lugar. Por ejemplo, si se tiene $\phi(p_1) = \{(\dot{x}_1 = 1), (x_1 \leq 6)\}$, cuando p_1 sea marcado, el cronómetro x_1 se activará y mientras su valor sea menor o igual a 6 la marca en p_1 permanecerá disponible y las transiciones en el $p_1 \bullet$ podrán ser habilitadas. Una vez que el cronómetro x_1 rebase el valor de 6, la marca dejará de estar disponible.

Se supone que todos los cronómetros están inicialmente detenidos y que su valor es 0, a menos que se especifique explícitamente lo contrario en algún lugar marcado. Una vez que el estado del cronómetro es declarado en un lugar marcado (activo/suspendido), éste continúa en el mismo estado a menos de que sea explícitamente indicado en algún otro lugar, es decir, un cronómetro cambia su estado solo cuando esto es indicado en un lugar marcado.

La función G declara algunas restricciones (guardas) al disparo de las transiciones, es decir, indica las condiciones adecuadas para que la transición asociada pueda dispararse. Por ejemplo, si se tiene: $G(t_1) = \{(x_1 == 6)\}$ significa que, cuando el marcado de la red habilite la transición t_1 y la guarda $(x_1 == 6)$ sea verdadera, entonces la transición t_1 puede ser disparada. Si la guarda en una transición es omitida, entonces la transición puede dispararse en cuanto el marcado disponible sea suficiente para habilitar la transición, es decir, su disparo depende únicamente del marcado de la red (y de la disponibilidad de dichas marcas) y no de las guardas en las transiciones. A continuación se indican las reglas de habilitación y disparo para este tipo de redes.

4.1.3 Reglas de habilitación y disparo

Definición. Una transición $t_j \in T$ se dice que está habilitada respecto a un marcado M ssi $\forall\, p_i \in P,\ M(p_i) \geq Pre(p_i, t_j)$, los invariantes en $\phi(p_i)\ \forall\, p_i \in \bullet t_j$ son verdaderos y $G(t_j)$ es verdadera.

Por ejemplo en la Figura 4.7.a, t_1 es habilitada en el marcado inicial (cuando el invariante es omitido, no hay restricción temporal para mantener la transición habilitada; de definirse algún invariante, éste debe de cumplirse para habilitar la transición correspondiente).

- ✓ Si el intervalo de tiempo definido por la guarda asociada a una transición t_j no se sobrepone con el tiempo inducido por los invariantes en los lugares en $\bullet t_j$ entonces t_j nunca será disparada. Por ejemplo, si se tiene una transición t_1 de una GSWPN (ver Figura 4.8) donde el $\bullet t_1 = \{p_1\}$ y luego si se define $\phi(p_1) = \{(\dot{x}_1 = 1), (x_1 \leq 6)\}$ y $G(t_1) = \{(x_1 > 6)\}$, entonces, por un lado $\phi(p_1)$ nos indica que la marca en p_1 permanece disponible únicamente mientras $(x_1 \leq 6)$, pero $G(t_1)$ indica que la transición solo puede habilitarse cuando $(x_1 > 6)$, por lo tanto, la transición t_1 no llegará a habilitarse ni a ser disparada, ya que la condición expresada en $G(t_1)$ se cumple cuando la marca en p_1 ya no está disponible para habilitar t_1.

- ✓ Si el invariante en un lugar p_i nunca se cumple (es decir, no llega a ser verdadero) o deja de ser verdadero debido al transcurso del tiempo, entonces las transiciones de salida a ese lugar ($p_i\bullet$) no podrán habilitarse (a menos claro que en alguna evolución de la red, el valor de los cronómetros involucrados sean modificados apropiadamente).
- ✓ Si t_j está habilitada, entonces t_j puede ser disparada. Por ejemplo, en la Figura 4.7.a si se supone que P2 y P3 están marcados y que $(x_2 = ttl)$ entonces, t_3 está habilitada y por lo tanto puede ser disparada ya que el valor de x_2 es precisamente el requerido para hacer la guarda en t_3 verdadera.
- ✓ Cuando una transición t_j es disparada, los cronómetros expresados en $INI(t_j)$ deben ser inicializados. Por ejemplo, en la Figura 4.7.a, cuando t_3 llegue a dispararse, el cronómetro x_2 será reinicializado a 0.
- ✓ El disparo de las transiciones se realiza de manera instantánea.

Considerando el modelo de la Figura 4.7.a, cuando t_3 es disparada, entonces el cronómetro x_1 es suspendido; éste se activará nuevamente después del disparo de la transición t_4; una vez que x_1 se activa nuevamente éste regresa con el mismo valor que tenía antes de ser suspendido (Figura 4.7.b). De esta forma, cada cronómetro es inicializado solo en el marcado inicial o cuando se especifique explícitamente en el disparo de una transición.

Puede observarse en la misma Figura 4.7.a, que se ha introducido un tipo especial de lugares *P5* y *P6* los cuales se han llamado *lugares de comunicación*; éstos se incluyen para representar el intercambio de mensajes entre agentes. De esta forma a cada lugar etiquetado como mensaje de entrada en una red, le corresponde un lugar etiquetado como mensaje de salida en otra red (ver Figura 4.9). Tener esta pareja de redes es equivalente a tener solo una red obtenida mediante la fusión de los *lugares de comunicación*. En la Figura 4.9, la transición dibujada con línea punteada puede utilizarse para modelar la pérdida de mensajes.

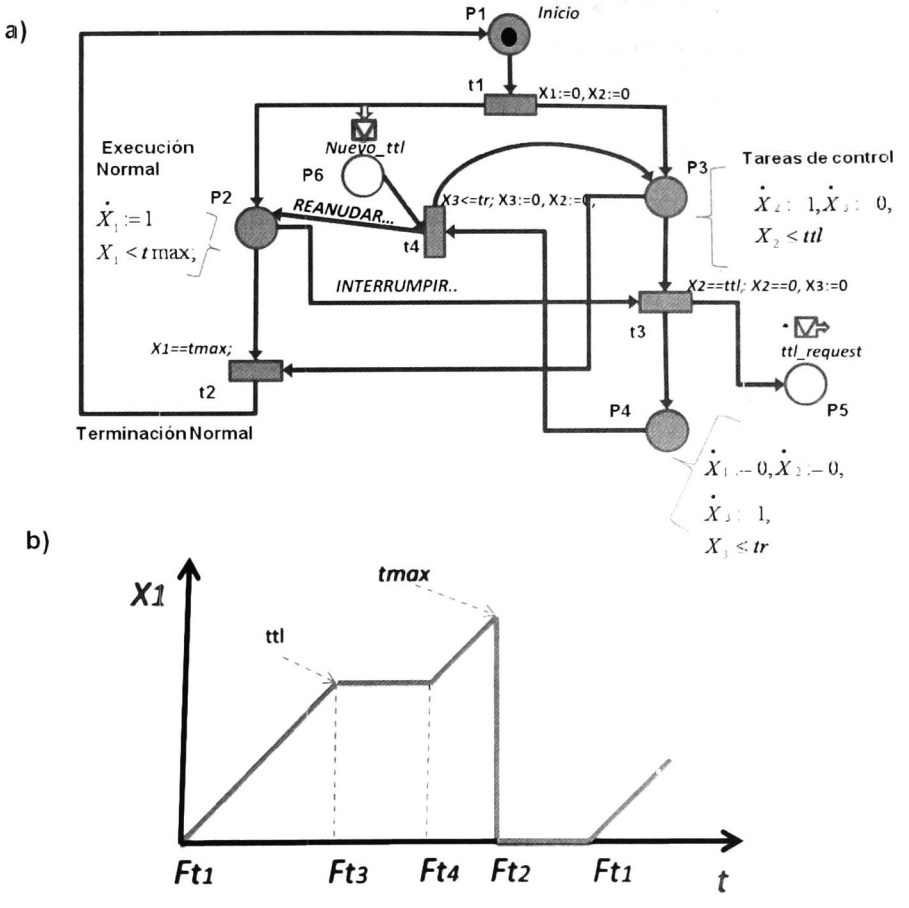


Figura 4.7 a) ejemplo de una GSWPN, b) escala de tiempo del cronómetro de acuerdo a algunos disparos de transiciones ( ).

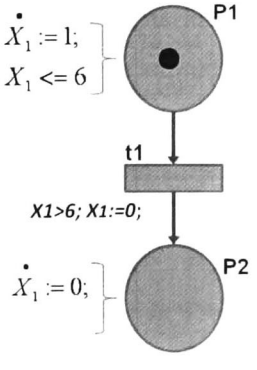


Figura 4.8 Ejemplo de una GSWPN donde, debido a los valores definidos del invariante en P1 y de la guarda en t1, la transición t1 jamás es habilitada

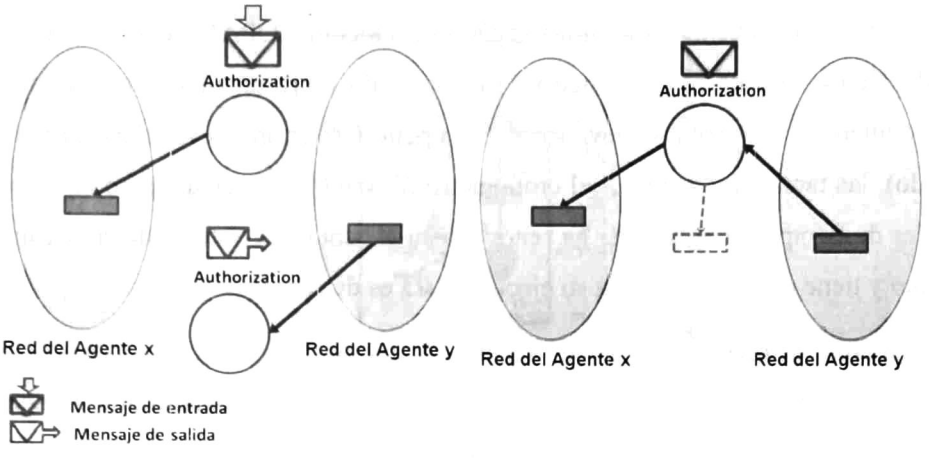


Figura 4.9 Representación de lugares para el intercambio de mensajes y su equivalencia

4.1.4 Modelado de un protocolo utilizando GSWPN

Se considera como caso de estudio el protocolo de control de agentes definido en (Padilla-Duarte, 2007) para mostrar el uso de las GSWPN. En este protocolo el agente se bloquea cuando su tiempo de vida (*ttl*) se agota y es restablecido a sus tareas normales cuando el *ttl* es renovado.

Tres de los componentes de este protocolo COPAM son presentados. En la Figura 4.10, el comportamiento de la entidad llamada *Agente Ejecutor* es descrito. En la parte izquierda de la red se modelan algunas de las tareas que forman el comportamiento ***normal*** del agente, o las tareas para las cuales el agente fue creado. La parte derecha de la red modela la parte de control del agente, es decir, las tareas encargadas a renovar su *ttl* según lo indicado en el protocolo COPAM. De esta forma cuando la transición *t1* en la red de la Figura 4.10 es disparada, los dos distintos comportamientos del agente son habilitados. Si *P2* es marcado, entonces las tareas normales del agente pueden ejecutarse y durante ese tiempo (modelado por el cronómetro X_1) el agente consume su *ttl*; mientras *t3* no sea habilitada (la transición se habilita cuando el *ttl* del agente es consumido X_2==*ttl*), el agente puede continuar su ejecución normal.

Si *t3* se habilita, es decir, cuando el *ttl* del agente es consumido, la actividad en *P2* es interrumpida (el agente se bloquea según el protocolo COPAM) después de haber sido iniciada, de esta forma el cronómetro X_1 es detenido representando esta característica de bloqueo mientras el agente renueva su *ttl*. Si la petición por un nuevo *ttl* es aprobada (*P7* es marcado), las tareas del agente y el cronómetro X_1 son restablecidas; pero si después de *tr* unidades de tiempo, el agente no ha renovado su *ttl*, entonces éste se declara como agente *huérfano* y tiene que terminar con su ejecución (*t5* es disparada).

Debido a que se modelan agentes reactivos, se incluyen varios lugares de comunicación en el modelo. Estos lugares corresponden a los incluidos, también como *lugares de comunicación*, en el modelo del Agente Pizarra del protocolo COPAM mostrado en la Figura 4.11. Hay que notar que el modelo del Agente Pizarra es enteramente reactivo así que el uso de cronómetros puede ser omitido.

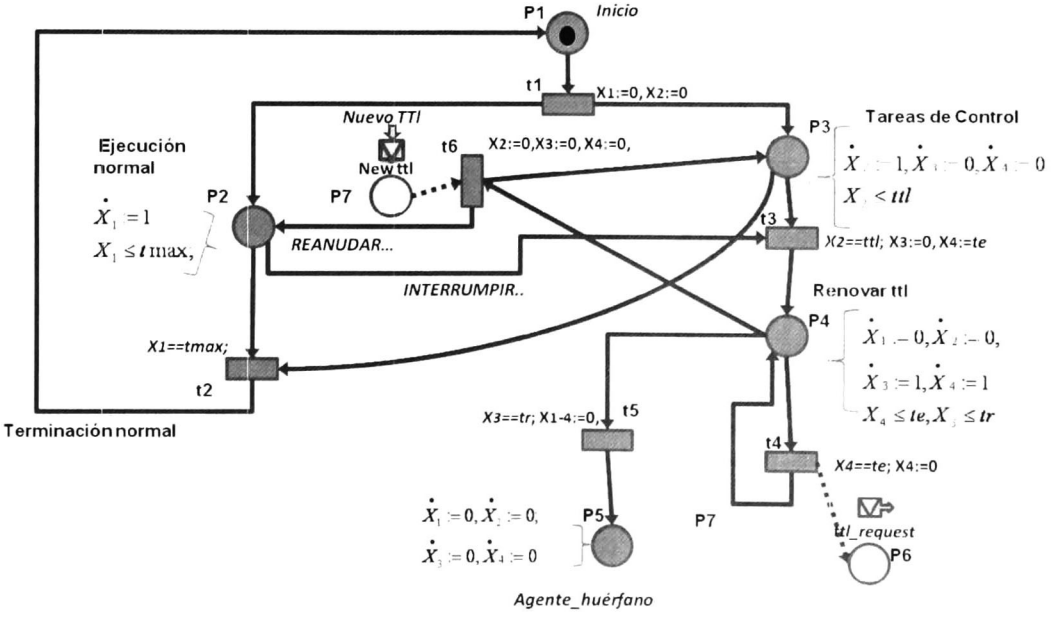


Figura 4.10. GSWPN para el agente Ejecutor del protocolo COPAM

Finalmente, el agente Controlador se modela con la red de la Figura 4.12 cuyo comportamiento es muy similar al mostrado por el Agente Ejecutor en la Figura 4.10.

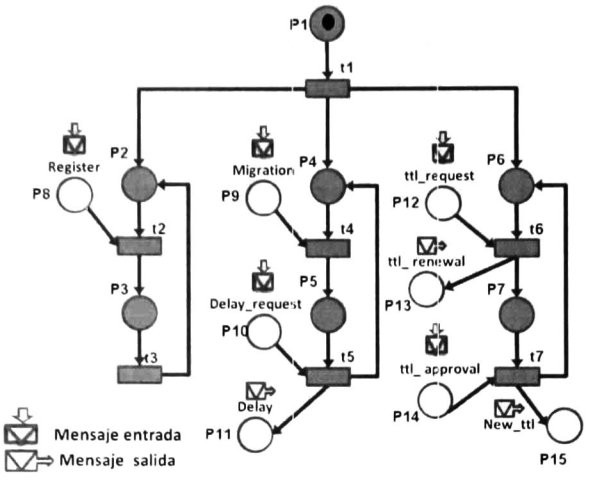


Figura 4.11. GSWPN para el Agente Pizarra del protocolo COPAM

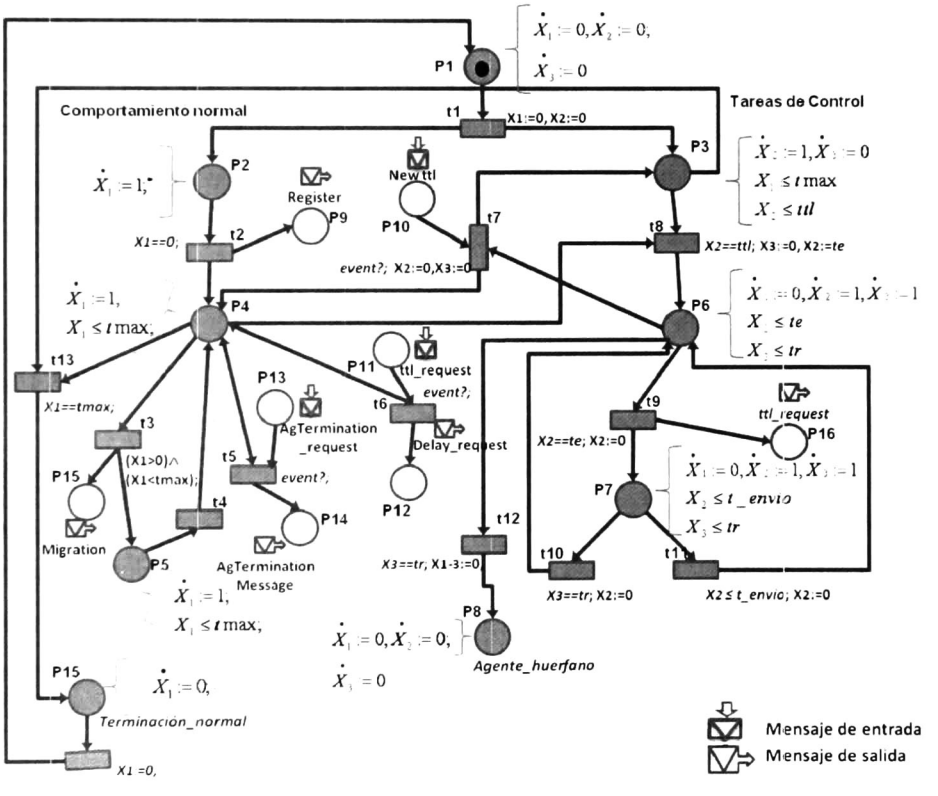


Figura 4.12. GSWPN del agente Controlador del Protocolo COPAM

Con la ayuda de las GSWPN se ha obtenido un modelo formal del protocolo COPAM (Padilla-Duarte, 2007), éste nos brinda las ventajas siguientes:

- Los elementos del modelo pueden ser descritos de manera independiente, haciendo más clara la función de cada uno de ellos.
- Debido a la naturaleza gráfica en el modelo, la función e interacción de los elementos del protocolo pueden ser comprendidos con mayor claridad.
- El proceso de cada elemento puede ser mejorado debido a la atención que se le está brindando en la descripción de los estados del mismo.
- Se puede observar y estudiar la evolución de los elementos en los modelos y por lo tanto, se podrían detectar posibles errores de planteamiento o de funcionamiento del protocolo, además se podrían hacer las modificaciones necesarias al mismo (de ser requeridas).
- Al incorporar el tiempo en los lugares, se puede obtener un análisis temporizado de las variables con los valores asignados para ver si cumplen los objetivos que el protocolo pretende, o si se llega a un estado deseado. Por ejemplo, se podría analizar en el protocolo COPAM, si con los valores de *ttl, tr* y *te,* el Agente Ejecutor puede llegar a terminar su ejecución de manera normal o se pasa a un estado de *Agente_huérfano*, aún cuando éste no lo sea.

4.2 Modelado de Protocolos de Control de Agentes

En esta sección se presentan los modelos de algunos de los protocolos propuestos en el capítulo 2, mostrando cada proceso separado en casos para una mayor comprensión de los mismos. Además se incluyen modelos de los comportamientos para los 4 diferentes tipos de entidades que interactúan en las descripciones de los protocolos.

Entre los protocolos de control de Agentes se tienen: Localización de Agentes, Terminación de Agentes y Control de Agentes huérfanos. Los modelos para cada una de las funcionalidades definidas se describen a continuación.

4.2.1 Localización de Agentes

La localización de agentes, como se describió en el capítulo 2, presenta dos casos importantes: 1) cuando el Agente Móvil se encuentra dentro de la misma zona (red) donde fue iniciado, es decir, dentro de la zona donde está su aplicación propietaria (localización interna) y 2) cuando se encuentra ejecutándose en una zona distinta a donde fue iniciado (localización externa).

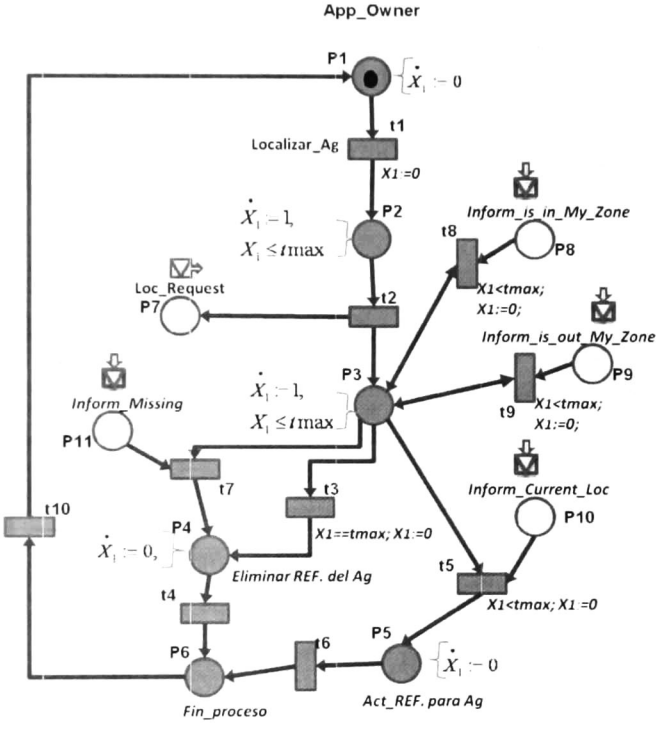


Figura 4.13 GSWPN para la Aplicación Propietaria, modelando el protocolo de Localización de un Agente Móvil

Localización interna

Para mostrar cómo sería el modelado de los agentes utilizando GSWPN, se supondrá que la aplicación propietaria del Agente móvil desea localizarlo; por tanto, es quien inicia el proceso de localización, enviando un mensaje de *Loc_Request* al Agente Administrador (**AM**) de la red actual. En la Figura 4.13 se muestra la GSWPN para la aplicación propietaria; puede observarse que después del disparo de *t2* se envía el mensaje de salida con la petición correspondiente. Después el agente pasa a un estado (P3) donde

espera (tarea de espera modelada por el cronómetro) por la información de la ubicación actual del agente (*Inform_Current_Loc* en P10). Si el tiempo de espera se agota ( ) o recibe un mensaje de *Inform_missing* (indicando que no se pudo localizar el agente), la aplicación pasa al estado de *Eliminar Referencia del Agente* P4. Los mensajes de entrada de *inform_is_in_my_zone* (P8) e *Inform_is_out_My_Zone* (P9) solo hacen que la aplicación incremente el tiempo de espera del mensaje con la ubicación del agente; el primer mensaje indica que el agente se encuentra en la red actual y el segundo que el Agente ha migrado a la red de una compañía diferente.

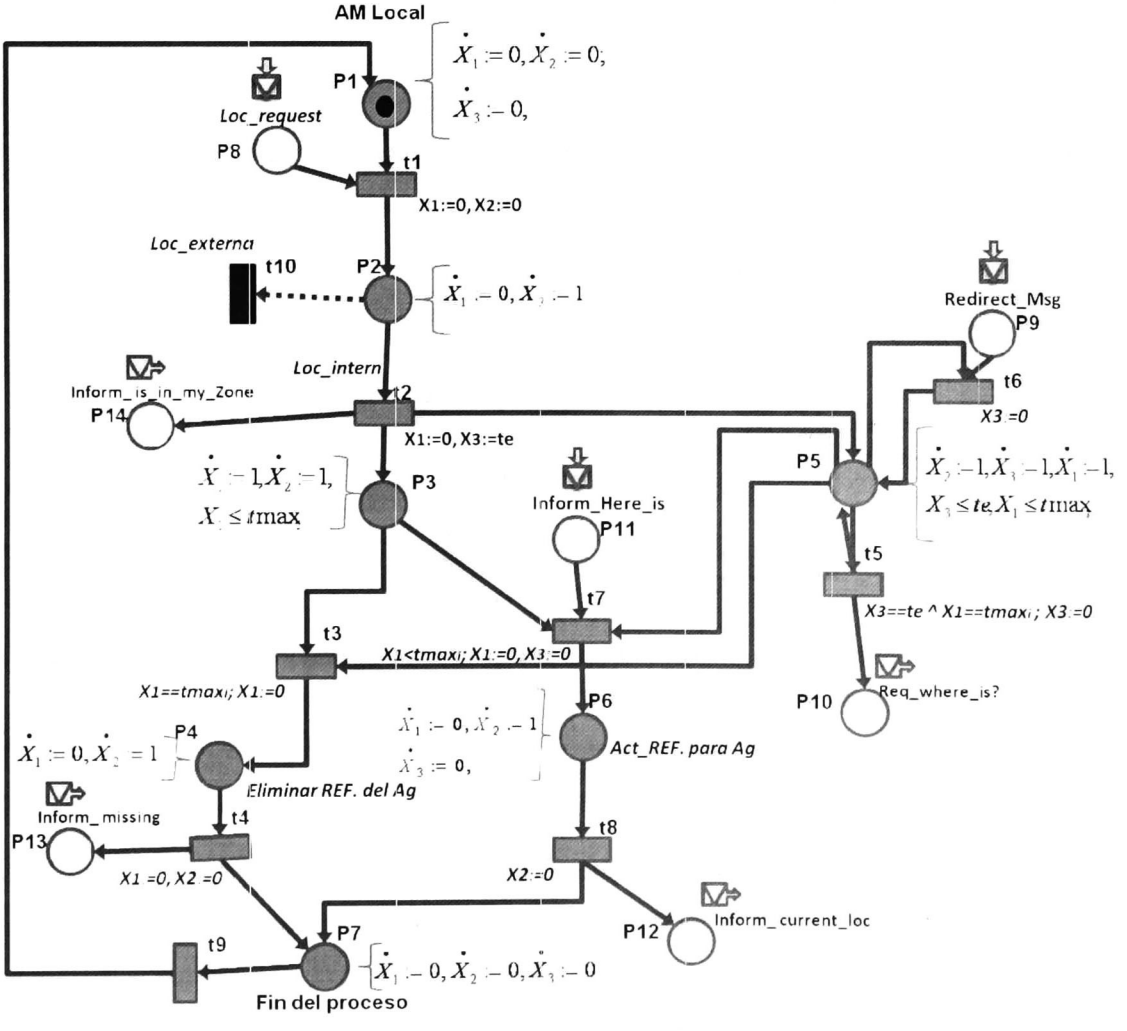


Figura 4.14 Fragmento de una GSWPN para el AM modelando el protocolo de Localización Local de un Agente Móvil

En la Figura 4.14 se muestra la GSWPN modelando parte del proceso de localización que coordina el **AM** donde, al recibir el mensaje *Loc_Request* (P8) que envió la Aplicación propietaria del Agente Móvil, el **AM** determina si el agente se encuentra en ejecución dentro de su zona (*Loc_intern* t2*)* o en una zona diferente (*Loc_externa* t10). En esta figura, se muestra solo un fragmento de la GSWPN correspondiente al proceso de localización interna, el fragmento restante (localización externa) es mostrado en la Figura 4.16. Cuando el **AM** determina que el agente móvil está dentro de la zona que maneja, se lo informa al agente que le hizo la solicitud (P14 *Inform_is_in_my_zone*); después, envía un mensaje al **AMLog** del último lugar (nodo/sitio) conocido del Agente Móvil (*Req_where_is?* P10) y espera (modelado por el cronómetro X_1) hasta un tiempo máximo ($X_1 == \text{tmax}_i$) para encontrar al Agente Móvil. Si el tiempo de espera se cumple, informa que el agente está perdido, es decir, que no pudo ser localizado dentro del tiempo establecido (P13 *Inform_missing*). Puede eliminar las referencias hacia dicho agente. Por otro lado, si recibe un mensaje *Inform_here_is* (P11), dicho mensaje contendrá información sobre el último sitio donde el agente se encuentra en ejecución, por lo que el **AM** actualizará dicha información en sus registros e informará de la ubicación del agente móvil (P12) a la entidad que le solicitó dicha información.

Otra entidad que participa en el proceso es el **AMLog**, es decir, el agente que se encuentra en cada nodo y registra el arribo/salida de los agentes móviles a dicho nodo. Otra función importante del **AMLog** es guardar en almacenamiento confiable toda operación que se realice en el sitio. En el protocolo de Localización (ver Figura 4.15) el **AMLog** recibe una petición *Req_where_is?* (P4) para que informe si el agente móvil a localizar se encuentra en ejecución dentro de ese sitio. Si el agente se encuentra en ese sitio, el **AMLog** envía un mensaje (P3) *Inform_Here_is* con la ubicación del agente. Si no se encuentra, envía un mensaje *Redirect* P5, informando que el agente ya abandonó ese sitio y que la petición se redireccionará *(Fwd:Req_where_is?* P6) al último sitio conocido del Agente móvil.

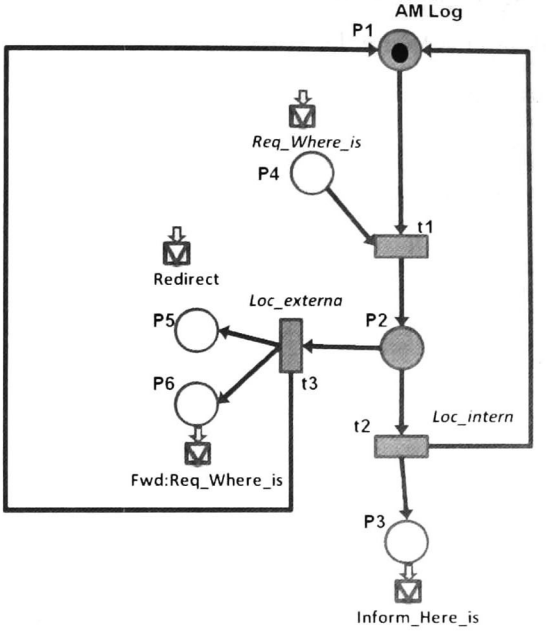


Figura 4.15 GSWPN del AMLog para el protocolo de Localización

Localización externa

En la localización externa se trata de encontrar primero la Red (compañía), donde el agente móvil se encuentra en ejecución y segundo, el sitio (o nodo) dentro de esa red donde el agente se encuentra. El proceso de localización externa por parte del **AM** se muestra en la GSWPN de la Figura 4.16. En esta Red se muestra que, cuando el **AM** determina que el Agente ha migrado a una red distinta (disparo de *t10)*, dicho agente tiene que comunicarse con un **AM** que administra la red de la zona donde se conoce que el Agente Móvil ha migrado. Una vez que se dispara *t10*, el **AM** revisa si el Agente Móvil que se desea localizar es un Agente que se inició dentro de esa red (que su aplicación propietaria está dentro de la red que el **AM** administra y por tanto es un *agente local*), o es un Agente Móvil externo (creado en otra red). Si es un *agente local* entonces el **AM** debe administrar el proceso de localización (disparo de *t11*), de lo contrario simplemente reenviará el mensaje de *Loc_Request* (*P21*) al **AM** de la última zona conocida donde migró el Agente móvil.

Como se muestra en la Figura 4.16, cuando P15 es marcado, significa que el AM está en la espera del mensaje que le informe la localización del Agente Móvil. Cuando este tiempo de espera se agota y no ha recibido esta información, el **AM** está listo para eliminar las referencias hacia dicho agente móvil (P16) y notificar a su aplicación propietaria que el agente no puede ser localizado (P23). Otra acción que puede tomar el **AM** es iniciar el proceso de recuperación del Agente Móvil en lugar de proceder a eliminarlo.

Si dentro del tiempo de espera (modelado por el cronómetro) el **AM** recibe un mensaje de otro **AM** informando que el agente se encuentra en ejecución dentro de su zona (*inform_is_in_my_Zone P24),* el **AM** espera un tiempo extra por la información del nodo (*P27 Inform_Current_Loc*) donde está actualmente el Agente Móvil. Una vez recibida esta información, el **AM** notifica a la Aplicación propietaria la ubicación del Agente Móvil (*P28 Inform_Current_Loc*).

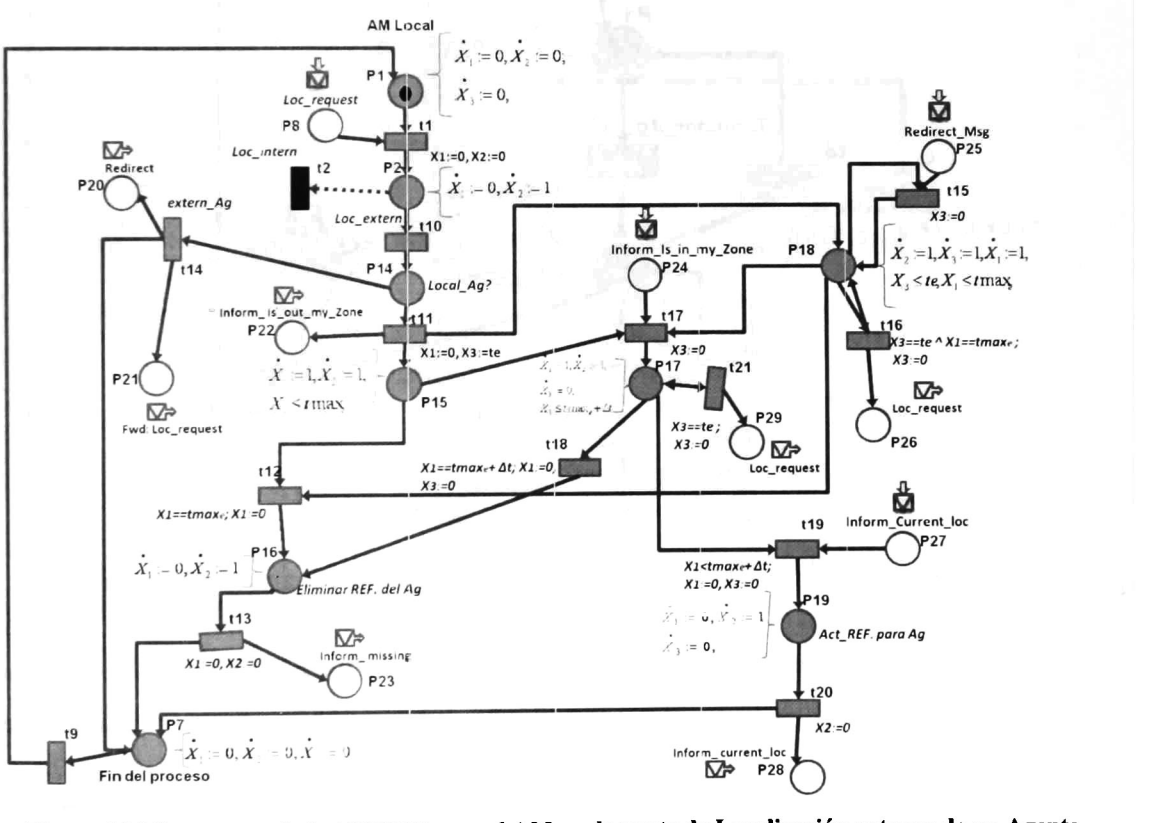


Figura 4.16 Fragmento de la GSWPN para el AM en la parte de Localización externa de un Agente Móvil

4.2.2 Terminación de Agentes

En los modelos en GSWPN de las entidades involucradas en el protocolo de terminación de Agentes, se supone que la Aplicación propietaria del Agente Móvil que se desea terminar, es quien inicia la petición por eliminación del Agente.

La GSWPN para el protocolo de terminación de Agentes perteneciente a la Aplicación propietaria, es la mostrada en la Figura 4.17, donde la Aplicación envía un mensaje (P4) para la terminación de un Agente Móvil al **AM** asociado a dicho Agente (**AM** de la red o zona Local); en esta situación espera la llegada de un mensaje de confirmación de dicha petición (*ACK_Termina P3*).

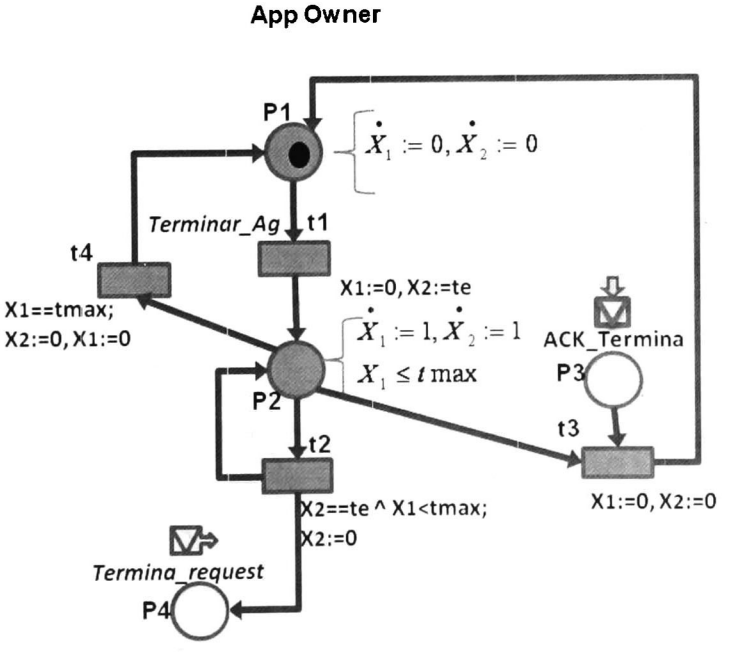


Figura 4.17 GSWPN para la Aplicación que inicia el proceso de Terminación de Agentes

Cuando un **AM** recibe un mensaje *Termina_request* (*P6* en la Figura 4.18), primero envía un mensaje *ACK_termina* (*P7*) a la entidad que le envió la petición (indicando que ha recibido el mensaje de terminación del Agente). Después determina si el agente móvil se encuentra en ejecución dentro del sitio actual. Si el Agente Móvil esta dentro del mismo nodo (disparo de *t7*) entonces se elimina toda referencia del agente en dicho nodo y se termina al Agente. Si el agente no se encuentra en el mismo nodo (disparo de *t3*), se reenvía

la petición al **AM** (o al **AMLog**) del último sitio conocido del Agente Móvil (*Fwd:Termina_request P8*). En esta situación espera la confirmación para eliminar las referencias a dicho agente. Tanto **AM** como **AMLog** tienen modelos de red similares.

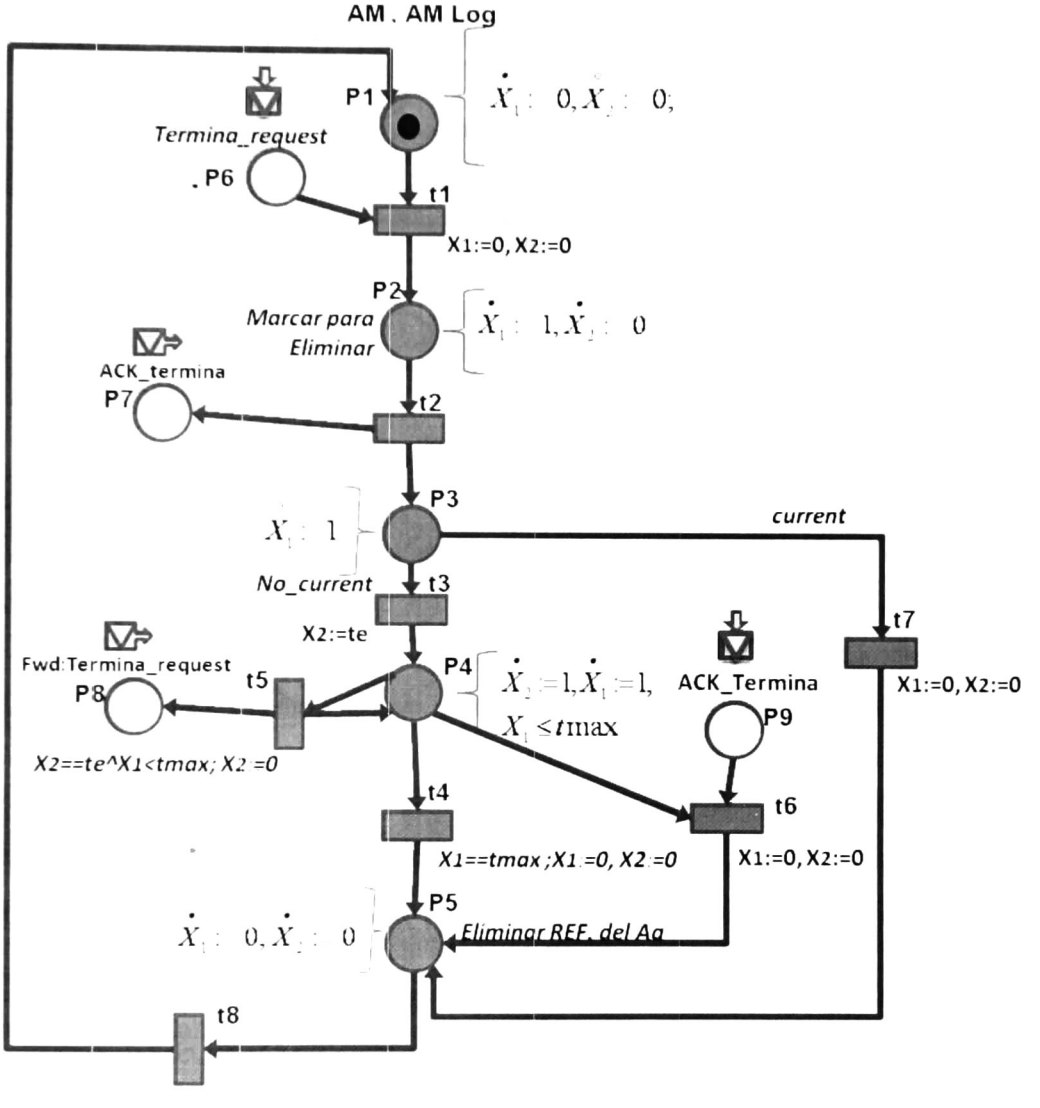


Figura 4.18 GSWPN para un AM y un AMLog modelando el protocolo de terminación de Agentes

4.2.3 Control de Agentes Huérfanos

Como se explicó en el capítulo 2, el control de Agentes huérfano está basado en el concepto de energía. En la asignación de un tiempo específico (tiempo de vida, permiso de ejecución) a un Agente Móvil durante su creación (por parte del **AM** de la red local) para que termine su ejecución. Una vez que dicho tiempo se cumple, el Agente Móvil suspenderá la ejecución de las tareas que se encuentre realizando e iniciará la renovación del tiempo de vida (*ttl*). Tanto el Agente Móvil, como su manejador (el **AM** asociado al agente) poseen esta información, por lo que pueden presentarse dos situaciones:

1) Que el Agente Móvil inicie el proceso de renovación de *ttl* enviando un mensaje al **AM** de la red donde se encuentra.

   En la Figura 4.19 se describe el comportamiento general de un Agente Móvil el cual, al dispararse *t1* inicia su comportamiento normal (la realización de las tareas para las cuales fue programado, modeladas por el cronómetro X_1) en P2 y P4. Al marcarse P3 se inicia un cronómetro (X_2) modelando el tiempo que el agente ha consumido de *ttl*. Cuando *t8* es disparado, significa que *ttl* del agente se ha agotado y las tareas normales del agente deben ser suspendidas (se toma la marca en P4 y P6, además se detiene el cronómetro X_1) mientras se realiza el proceso de *renovación de tiempo de vida* (para lo cual envía un mensaje de *New_ttl_request* al AM local).

   Cuando el Agente móvil recibe un mensaje *New_ttl*, se restaura su comportamiento normal (regresa la marca a P4 y reanuda el cronómetro X_1) y se reinicia el cronómetro que controla el *ttl* ($X_2 := 0$ en *t7*). Si el tiempo para renovar su *ttl* se agota, *t12* es disparado y el agente móvil se considera como *agente huérfano* (se marca P8). En esta situación el agente móvil se eliminaría para evitar gastar recursos de manera innecesaria.

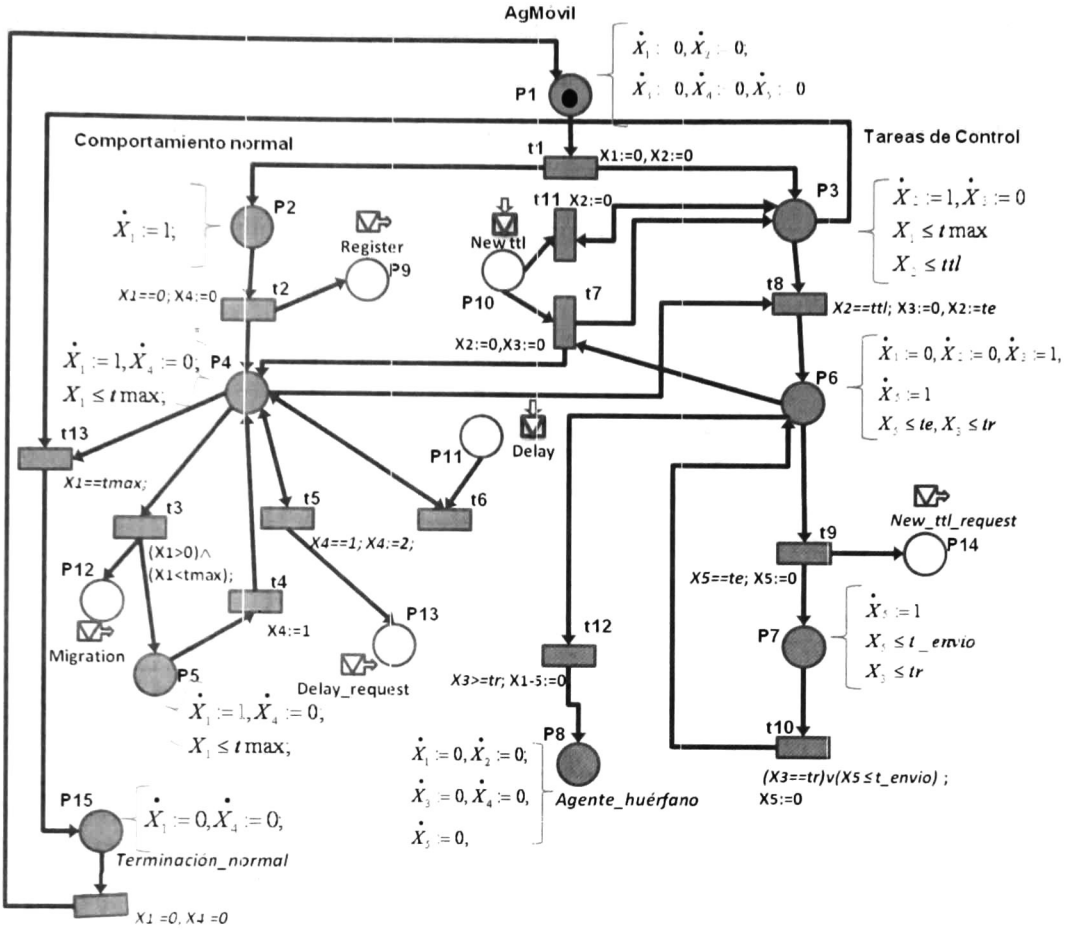


Figura 4.19 Modelo de un Agente móvil mostrando el proceso de control de tiempo de vida utilizando GSWPN

Cuando el **AM** local recibe un mensaje de *New_ttl_request* (modelado en la Figura 4.20 al marcarse P6), primero determina si el agente móvil que realiza la petición es un agente asociado a él, es decir, un agente que fue iniciado en su zona por una aplicción local y por tanto es responsable de su administración (agente local). Si se trata de un Agente local el **AM** envía un mensaje *New_ttl_request* (P7) a la aplicación propietaria (modelada por la red de la Figura 4.21) del agente para que confirme si éste aún es necesario o no. Si recibe un mensaje de *Reject_ttl* (P10) por parte de la aplicación propietaria, el **AM** deberá iniciar el proceso de terminación del Agente Móvil en cuestión. Si recibe un mensaje de *Renew_ttl* (P9), el **AM** actualiza las referencias hacia el Agente Móvil y autoriza la renovación mediante un mensaje *New_ttl* (P8).

Cuando se trata de un Agente Móvil que no es local, es decir, un agente que fue creado en una red distinta, el **AM** de la red actual tiene que decidir si otorga o no un nuevo permiso de ejecución. En la Figura 4.20 se describe el caso en que el **AM** de la red actual siempre otorga un nuevo *ttl* a los agentes que no son locales, pero podría modelarse un comportamiento diferente dependiendo de las políticas que posea cada **AM**.

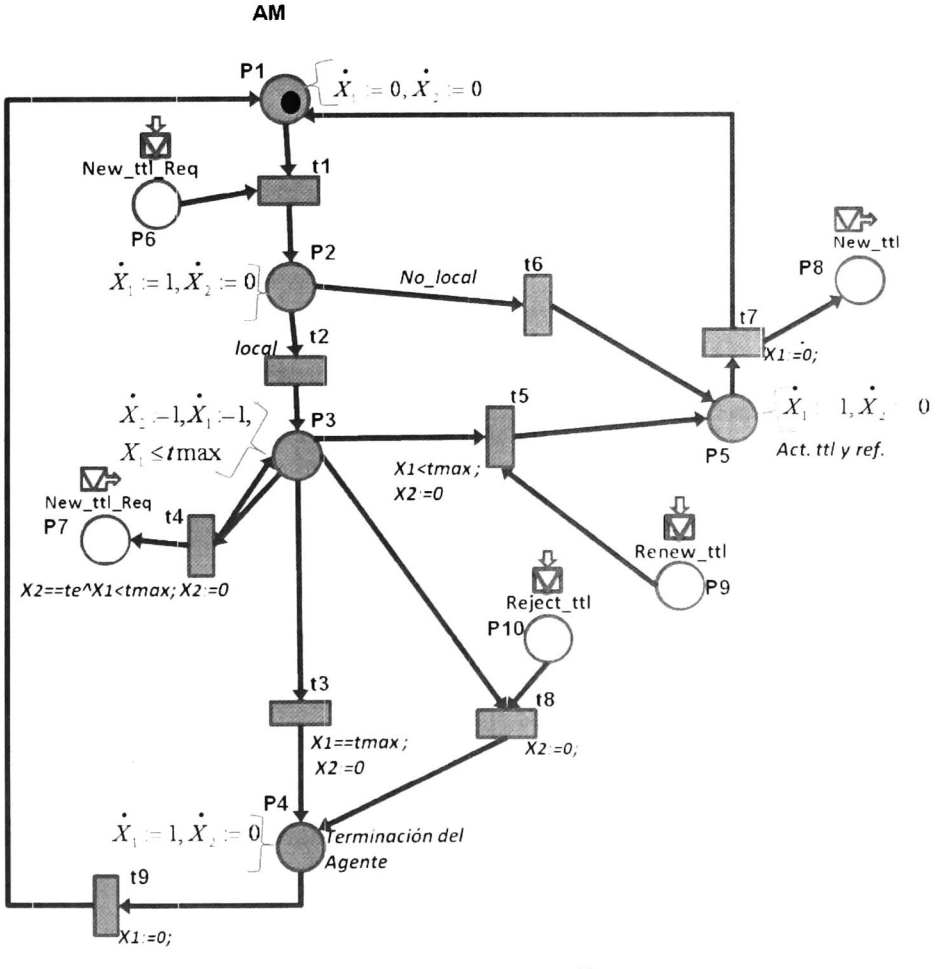


Figura 4.20 GSWPN de un AM para el proceso de Renovación de ttl iniciado por un Agente Móvil

En la Figura 4.21 se describe el comportamiento de Aplicación propietaria del Agente Móvil que inicia el proceso de renovación de *ttl*. La Aplicación al recibir un mensaje *New_ttl_Request* (P3), decide entre enviar un mensaje de *Renew_ttl* para aprobar la

renovación del tiempo de vida (P4) o un mensaje para rechazar dicha solicitud (*Reject_ttl* en P5).

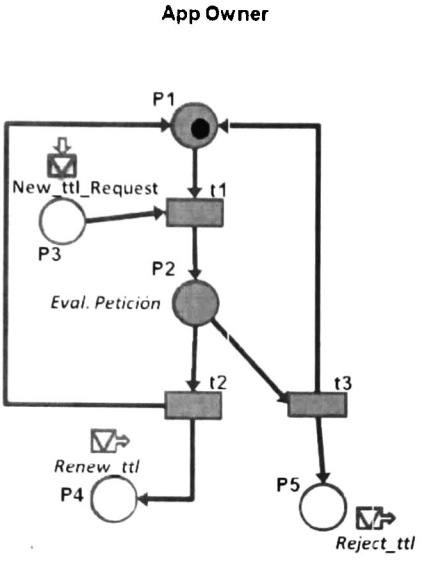


Figura 4.21 GSWPN para la Aplicación propietaria de un Agente Móvil que inicia su proceso de renovación de ttl

2) El **AM** determina que ya ha transcurrido el *ttl* del Agente móvil asociado y no ha recibido un mensaje de renovación, entonces inicia el proceso de localización del Agente para renovar así su *ttl*.

   En la Figura 4.22 se describe la GSWPN del **AM** que controla el *ttl* de cada Agente móvil, ya sea que se tenga una asociación directa en el momento de su creación (por medio de una aplicación perteneciente a su red local), o que sea un agente externo que entró a la zona para guiar una parte de un proceso de WF. Cada **AM** administra tanto a los agentes locales como a los agentes que vienen de otras zonas (Agentes externos) y que en ese momento se encuentran en ejecución dentro de su red local. El **AM** primero evalúa qué tipo de asociación existe con el Agente Móvil; si es un agente local se dispara *t2* y si es un agente externo se dispara *t3*.

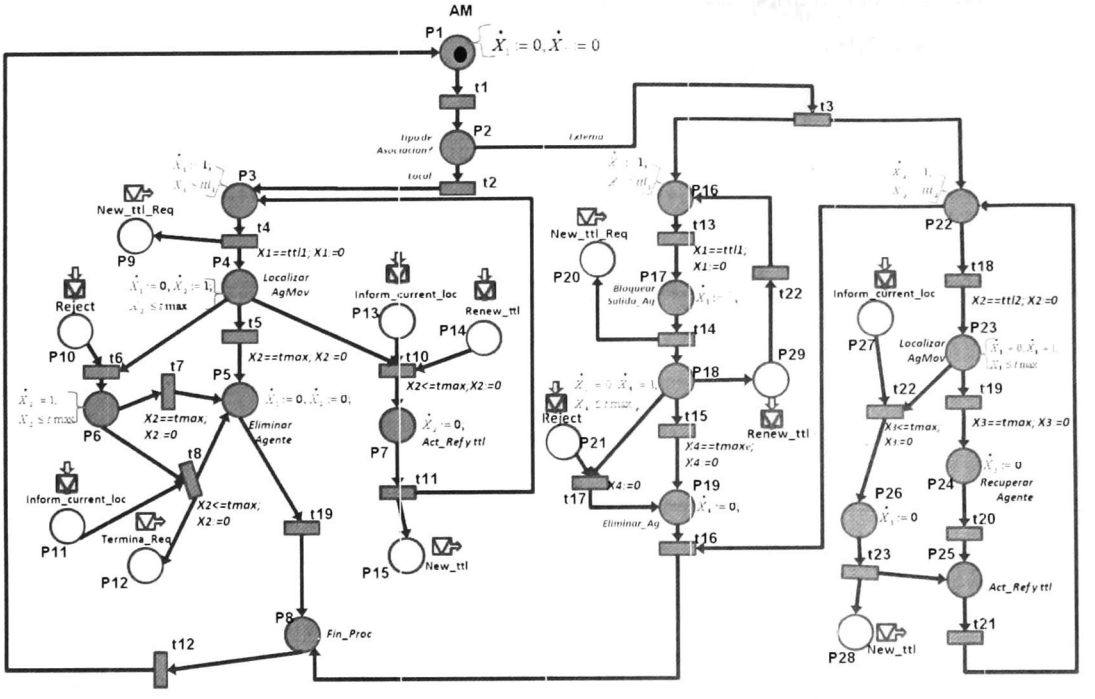


Figura 4.22 GSWPN de un AM que inicia el proceso de renovación de ttl de un Agente Móvil

Cuando se trata de un Agente local (se dispara *t2*), el **AM** envía un mensaje *New_ttl_request* (P9) a la aplicación propietaria para que se verifique si la ejecución del Agente Móvil aún es necesaria o no. Enseguida, el **AM** inicia el proceso de Localización del Agente Móvil (abstraído en P4 pero que fue descrito en las Figura 4.13 y Figura 4.14). Una vez que el **AM** recibe un mensaje de *Reject* (P10) por parte de la aplicación propietaria, significa que el agente móvil ya no es requerido y por lo tanto debe de terminar su ejecución. Si el **AM** recibe un mensaje de *Renew_ttl* (P14), significa que la aplicación autoriza la renovación del tiempo de vida del agente Móvil. En esta situación solo se espera a que el agente móvil sea localizado (que se marque el lugar P13 con la llegada del mensaje *Inform_current_Loc*) para poderle enviar un mensaje de autorización de *ttl* (P15). Sin embargo, si el tiempo para localizar al agente móvil se termina (cronómetro *X2==tmax*), se elimina toda referencia al mismo (P5).

Si se trata de un Agente externo (disparo de *t3* en la Figura 4.22), el *ttl* se maneja de manera especial. Cada **AM**, al llegar un agente móvil a la red que administra, asigna un nuevo *ttl* a dicho agente para que se ejecute dentro de su red y almacena el *ttl* que ya

tenía asignado el agente. De esta forma, **AM** registra ambos tiempos de vida: 1) el que le asignó el manejador del agente móvil (ttl_1) y 2) el nuevo ttl (ttl_2) que se le asigna al agente móvil al entrar a la red. Por lo tanto, el **AM** manejará dos cronómetros, uno para cada *ttl* del agente móvil. En la Figura 4.22, el ttl_1 se modela con el cronómetro X_1 en P16, y el ttl_2 asignado al Agente móvil al entrar a la red se modelada con el cronómetro X_2 en P22.

Cuando el ttl_1 se cumple, el **AM** de la red actual se encarga de comunicarse con el respectivo **AM** manejador para renovar el tiempo de vida del Agente móvil. En esta situación, envía un mensaje de renovación de *ttl* (mensaje *New_ttl_Req P20*) al **AM** propietario; después, bloquea el permiso de salida de la red para el Agente móvil respectivo (ya que si llegara un mensaje de *Reject* P21 significaría que el Agente Móvil ya no es requerido y que debe de eliminarse). Al recibir un mensaje de *Renew_ttl* (P29) actualiza el cronómetro respectivo.

Cuando el ttl_2 se termina, el **AM** de la red actual inicia el proceso de localización del Agente Móvil externo que se encuentra en su red (P23). Una vez que se recibe mensaje con su ubicación (*Inform_current_loc* P27), simplemente se renueva el permiso de ejecución local al Agente Móvil. Si el tiempo para localizar al Agente Móvil externo se termina (disparo de *t19*) el **AM** podría iniciar el proceso para la recuperación del Agente móvil.

Los modelos para el Agente Móvil y para la aplicación propietaria quedarían como los descritos por la Figura 4.19 y Figura 4.21 respectivamente.

Al utilizar GSWPN se ha modelado comportamiento de los protocolos propuestos en el Capítulo 2, mostrando así los tiempos involucrados en las tareas, así como el intercambio de mensajes entre las entidades involucradas en los protocolos.

Una vez obtenido el modelado de las entidades se podría estudiar su comportamiento mediante la simulación de los modelos obteniendo así un análisis temporizado según los parámetros asignados a cada protocolo (como se verá en el siguiente capítulo).

Capítulo 5

Análisis de protocolos para el control de agentes

Resumen. En este capítulo se analizan los protocolos propuestos mediante el análisis temporizado de los modelos en *Redes de Petri con Cronómetros Globales*. El análisis se realiza mediante la simulación de los modelos.

5.1. Análisis de modelos en Global StopWatch Petri Nets

A nuestro conocimiento no existe una herramienta para simular alguna versión de RP con cronómetros y más aun la extensión propuesta que manipula cronómetros globales (Global Stopwatch Petri Nets- GSWPN).

Dado que se incluyen en el modelo cronómetros que representan la ejecución de una tarea, sería de gran utilidad el poder obtener información del modelo, por ejemplo, obtener la región de alcanzabilidad de algún cronómetro, dado un marcado inicial de la red y así realizar un análisis temporizado del modelo capturado en GSWPN.

En esta tesis se muestran dos alternativas para hacer análisis temporizado de un modelo en GSWPN; la primera utilizando una herramienta llamada PHAVer (Frehse, 2006) (orientada a autómatas temporizados) comúnmente adoptada por autores que utilizan SWPN; y la segunda mediante la simulación del modelo.

5.2 Análisis mediante la herramienta PHAVer

En esta sección se presenta de manera general la herramienta de análisis PHAVer (Frehse, 2006) la cual es usada para el análisis de Autómatas lineales Híbridos. A falta de simuladores para SWPN, esta herramienta ha sido utilizada (Magnin et al., 2006a), transformando el modelo en SWPN a un autómata lineal híbrido equivalente.

5.2.1 Descripción de la Herramienta PHAVer

PHAVer (Polyhedral Hybrid Automaton Verifyer), de acuerdo a (Frehse, 2006) es una herramienta para el análisis temporizado de Autómatas Lineales Híbridos y posee las siguientes características:

- Aritmética exacta y robusta basada en una biblioteca para manejo de poliedros descrita en (Bagnara et al., 2002).
- Sobreaproximaciones en tiempo de simulación
- Pone restricciones en las computaciones basadas en poliedros.

Esta herramienta puede tratar varios tipos de problemas, por ejemplo: problemas de convergencia, explosión combinatoria y no determinismo. Utiliza una partición de lugares de acuerdo a restricciones especificadas por el usuario para limitar y localizar sobreaproximaciones. PHAVer trata con autómatas híbridos que pueden ser analizados utilizando poliedros, es decir, formulas lineales finitas. La herramienta está disponible bajo los sistemas operativos Linux y Windows, aunque la versión utilizada fue para Linux ya que pareció más estable.

Para utilizar esta herramienta normalmente se siguen los siguientes pasos:

1. Se escriben en un archivo (que se guardará con extensión ***.pha***) las partes que componen el autómata híbrido. Un ejemplo de la estructura que debe llevar un archivo de este tipo se describe en la **¡Error! No se encuentra el origen de la referencia.**.
2. Se escribe en otro archivo (aunque puede escribirse al final del archivo anterior) los comandos descritos en (Frehse, 2006) para obtener las regiones alcanzables mediante el forward/backward análisis. Se debe escribir explícitamente donde se guardará la salida, ya que PHAVer solo regresa un conjunto de puntos o vértices para ser graficados.
3. Dar como entrada ambos archivos al programa PHAVer por ejemplo:

   phaver sys.pha reach.pha.

4. Graficar la salida.

```
automaton aut
  contr var: var ident,var ident,.    ;
  input var: var ident,var ident,. . . ;
  parameter: var ident,var ident,. . . ;
  synclabs: lab ident,lab ident,.    ;
  loc loc ident:while invariant wait { derivative };
     when guard sync label ident do {trans rel } goto loc ident;
     when guard sync label ident goto loc ident;
     when . . .
  loc loc ident:  while . .
  initially: initial states;
end
```

Figura 5.1 Formato de archivo para describir a un Autómata Híbrido utilizado por PHAVer

5.2.2 Ejemplo de análisis de autómatas utilizando PHAVer

A continuación se mostrará un ejemplo sencillo de un autómata para mostrar el tipo de análisis que puede lograrse utilizando la herramienta PHAVer.

Ejemplo 1. Considere el autómata descrito por la Figura 5.3a, en el que se tienen dos localidades L1 y L2. Para pasar a la localidad L2 se requiere que el cronómetro modelado con x_1 sea igual a 10.5, además una vez en L2, el cronómetro x_1 estará detenido.

Tomando los datos de la Figura 5.3a, se puede describir el Autómata de entrada para el analizador PHAVer como lo descrito por la Figura 5.2.

Análisis hacia adelante

Mediante el o *Análisis hacia adelante* (Forward Analysis), se pueden determinar los estados alcanzables, de acuerdo a los cronómetros y estados iniciales definidos en la descripción del autómata. Los estados alcanzables para el ejemplo 1 se muestran en la Figura 5.3b, donde se grafica el cronómetro x_1 contra el cronómetro x_2 y pueden observarse los estados donde el cronómetro x_1 permanece detenido.

```
//System Parameters
b1 := 10.5;
c1 := 12;
//automaton definition
automaton sys
state_var : x1,x2;
synclabs: a;
loc L1: while 0<= x1 <= b1  wait { x1'==1 & x2'==1 };
        when  x1 == b1 sync a do { x1'==x1 & x2'==x2 } goto L2;
loc L2: while  x2 <= c1  wait {x1'==0 & x2'==1 };
initially: L1 & x1 == 0 & x2==0;
end
```

Figura 5.2 Descripción del Autómata para el Ejemplo 1

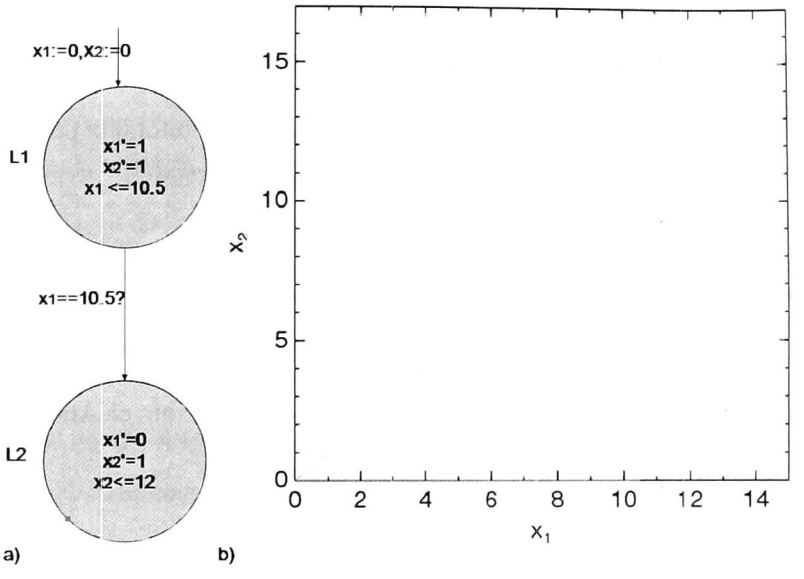


Figura 5.3 a) Autómata del ejemplo 1, b) regiones alcanzables para sus cronómetros (análisis hacia adelante)

Análisis hacia atrás

Para el análisis *hacia atrás* (backward analysis), se determinan los estados desde los cuales es posible llegar a un estado *deseado*. En PHAVer, el *análisis hacia atrás*, se obtiene invirtiendo el autómata, es decir, los arcos tienen dirección inversa, el tiempo disminuye en lugar de incrementarse, y el estado *deseado* del autómata se define como el estado inicial. En la Figura 5.4 se muestra un autómata (con dos cronómetros x_1 y y_2) cuyo estado inicial (*deseado*) se define como $x_1=y_2=10.5$; puede observarse en el autómata que la función de variación del tiempo en cada lugar se ha definido como -1, lo cual indica que el tiempo va a disminuir en lugar de incrementarse. La región azul definida en la gráfica indica cuales son los estados alcanzados de los cronómetros del autómata.

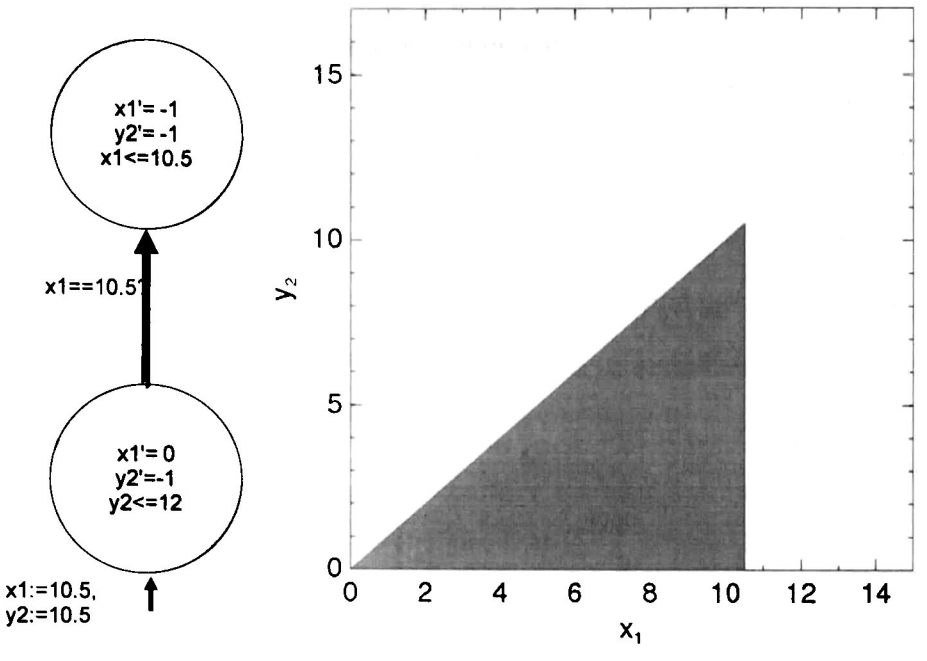


Figura 5.4 Análisis hacia atrás para el autómata del ejemplo 1

5.2.3 Análisis de un caso de estudio utilizando PHAVer

En esta sección se utiliza PHAVer para analizar las restricciones de tiempo de las tareas interactivas realizadas por los componentes del protocolo COPAM modelado con Global Stopwatch Petri Nets (GSWPN) en el capitulo anterior. Para ello se obtendrá primero el modelo de un autómata con cronómetros equivalente.

5.2.3.1 Autómata con cronómetros (StopWatch Automata-SWA)

La descripción del Autómata con cronómetros (SWA) utilizada en PHAVer, es similar a la descripción definida por Cassez en (Cassez & Roux, 2006), la cual es definida a continuación.

Definición 2. Un autómata con cronómetros es una 7-tupla (L, l_0, X, Σ, A, Inv, Dif) donde (ver Figura 5.5):

- **L,** es un conjunto finito de lugares (localidades)
- **l_0,** es el lugar inicial
- **X**, es un conjunto finito de relojes evaluados en los números reales
- **Σ,** es un conjunto finito de acciones
- $\boldsymbol{A \subset L \times C(X) \times \sigma \times 2^X \times L}$, es un conjunto finito de arcos. $a = (l, \delta, \sigma, R, l') \in A$ es el arco entre los lugares l y l', con la guarda δ, la acción σ y el conjunto de relojes a reinicializar R. $C(X)$ es el conjunto de restricciones sobre X.
- $\boldsymbol{Inv} \in \boldsymbol{C(X)^L}$, asigna un invariante a cada lugar
- $\boldsymbol{Dif} \in (\{\mathbf{0},\mathbf{1}\}^X)^L$, asigna una actividad a cada lugar, siendo $\dot{X}$ el conjunto de derivadas para los relojes. Por ejemplo dado un lugar l y un reloj x, se tiene que $\boldsymbol{Dif(l)(x)_{x \in X}} = \{\mathbf{0},\mathbf{1}\}$.

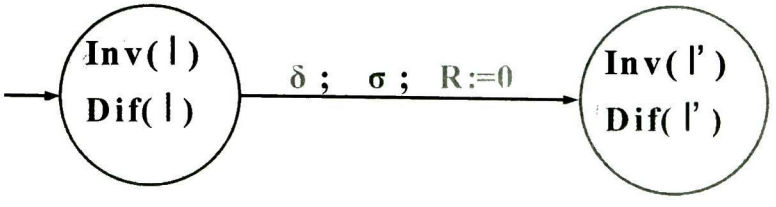


Figura 5.5 Partes principales de un Autómata con cronómetros

5.2.3.2 Construcción de un SWA a partir de una GSWPN

La obtención del SWA a partir de una GSWPN es simple. Primero se debe encontrar el Grafo de alcanzabilidad (G) de todos los posibles marcados de la GSWPN, definiendo un grafo **G= (M, A),** donde **M** es el conjunto de todos los posibles marcados del GSWPN: $M= \{M_0,....,M_i\}$ y **A** es el conjunto de arcos entre los nodos del grafo de marcados: $A=\{a_0,...,a_j\}$

Posteriormente el SWA se construye como sigue:

1. Tomar cada elemento del conjunto M, como un lugar l_i del SWA.

   a. Asociar a cada lugar l_i, las ecuaciones diferenciales que expresan la dinámica de los cronómetros en ese marcado y los invariantes. Para los invariantes se toma, por ejemplo para un marcado M_l, los invariantes de cada uno de los lugares con una marca y se colocan juntos en el lugar l_l equivalente.

   b. Un invariante es asociado con cada marcado M_k; así que por construcción, en cada marcado solo la posible evolución del tiempo es calculado, es decir, solo los cronómetros activos o suspendidos serán representados en cada lugar. Por lo tanto, debe agregarse cada invariante descrito en los lugares marcado en M_k en el lugar equivalente del Autómata l_k.

2. Las guardas de cada arco a_k del grafo G corresponden al disparo de una transición en la GSWPN, por lo que se asocian las guardas correspondientes de las transiciones de la GSWPN

Ahora se aplicarán lo pasos anteriores para obtener el SWA equivalente de la GSWPN para el agente Ejecutor del protocolo COPAM modelado en el capítulo anterior (ver Figura 4.10), sin embargo, para el análisis se obtuvo una red equivalente más sencilla (ver Figura 5.6) en la que se eliminan los lugares de comunicación con otras redes. Al aplicar los pasos para obtener el SWA equivalente, se obtiene el modelo de la Figura 5.7 en la cual se tienen 6 lugares de acuerdo a los 6 posibles marcados de la red de la Figura 5.6; además, se le agregó la información referente a los cronómetros involucrados en cada marcado.

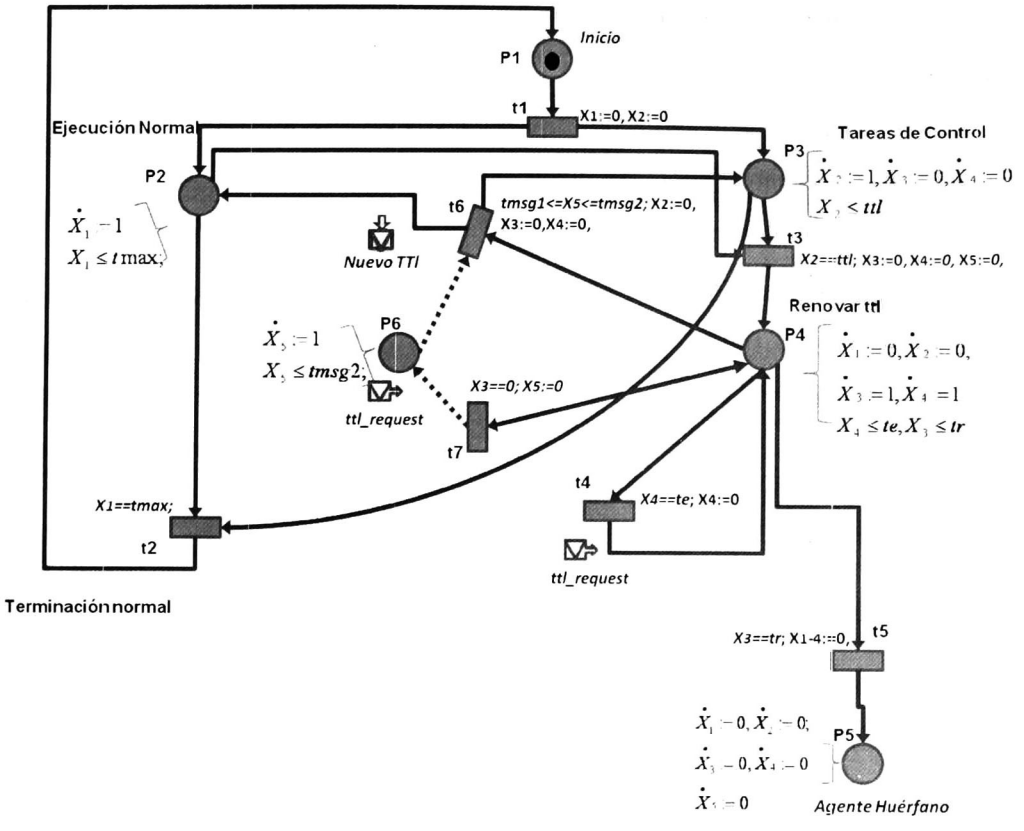


Figura 5.6 GSWPN reducida para el Agente Ejecutor del protocolo COPAM

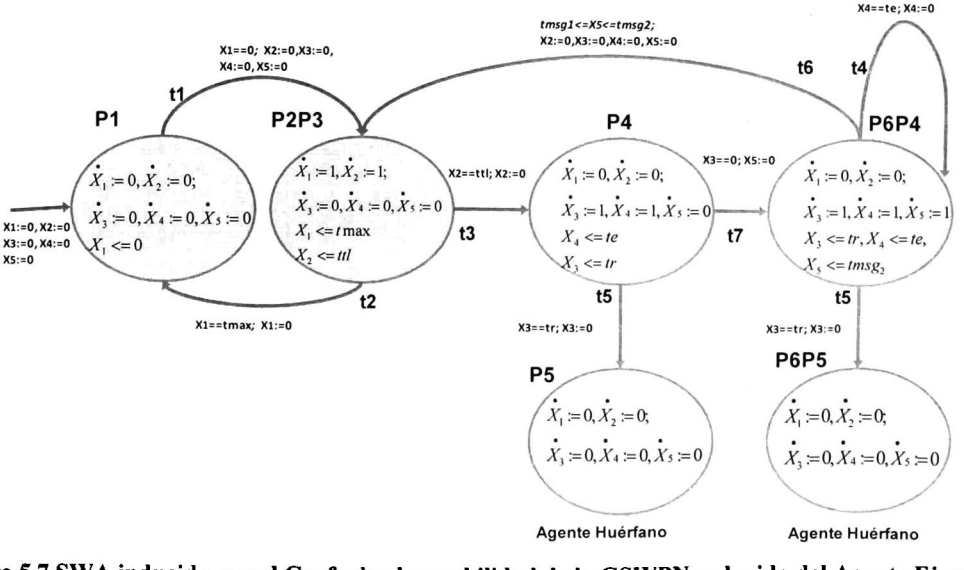


Figura 5.7 SWA inducido por el Grafo de alcanzabilidad de la GSWPN reducida del Agente Ejecutor del protocolo COPAM

Resultados

Una vez obtenido el autómata SWA, se escribe el archivo de descripción del Autómata, el cual será la entrada para el programa PHAVer. Al tomar como referencia la plantilla descrita en la Figura 5.1, se obtiene la descripción del autómata escrito en la Figura 5.8. Este autómata se pasará como entrada a PHAVer para encontrar las regiones alcanzables de los cronómetros mediante el *análisis hacia adelante.*

5.2.3.3 Análisis hacia adelante del caso de estudio

Para realizar el análisis, se asigna un valor determinado a cada uno de los parámetros que utiliza el protocolo y, con la variación de los mismos, se observará la variación de los estados alcanzados de los cronómetros del autómata. Los parámetros a modificar son:

- *tmax:* tiempo máximo aproximado de duración de las tareas normales del agente ejecutor, es decir, el tiempo total en ejecución del agente (sin contar el tiempo que el agente está bloqueado).
- *ttl:* tiempo de vida asignado al Agente.
- *te:* tiempo de espera
- *tr:* tiempo de reserva
- *tmsg1* y *tmsg2*: intervalo de probable tiempo de llegada de mensaje de aprobación de renovación de ttl.

```
//System Parameters
tmax := 15;
ttl := 5;
te := 1;
tr := 4;
tmsg1 := 3;
tmsg2 := 6;
//automaton definition
automaton sys
state_var : x1,x2,x3,x4,x5,x6; //x6 el tiempo normal
synclabs: a;

loc P1: while x1==0 wait { x1'==0 & x2'==0 & x3'==0 & x4'==0 & x5'==0 & x6'==0 };
  when x1==0 sync a do { x1'==0 & x2'==0 & x3'==0 & x4'==0 & x5'==0 & x6'==0} goto P2; //T1

loc P2: while 0 <= x1 <= tmax & 0 <= x2 <= ttl wait {x1'==1 & x2'==1 & x3'==0 & x4'==0 & x5'==0 &
x6'==1};
  when x1 == tmax sync a do {x1'==0 & x2'==0 & x3'==0 & x4'==0 & x5'==0 & x6'==x6} goto P1; //T2
  when x2 == ttl sync a do {x1'==x1 & x2'==x2 & x3'==0 & x4'==0 & x5'==0 & x6'==x6} goto P4; //T3

loc P4: while 0 <= x3 <= tr //& 0 <= x4 <tr wait {x1'==0 & x2'==0 & x3'==1 & x4'==1 & x5'==0 &
x6'==1};
  when x3 == tr sync a do {x1'==0 & x2'==0 & x3'==0 & x4'==0 & x5'==0 & x6'==0} goto P5; //T5 orphan
  when x4 == 0 sync a do {x1'==x1 & x2'==0 & x3'==x3 & x4'==0 & x5'==0 & x6'==x6} goto P6; //T4
sending request

loc P6: while 0 <= x3 <= tr wait {x1'==0 & x2'==0 & x3'==1 & x4'==0 & x5'==1 & x6'==1};
  when x3 == tr sync a do {x1'==0 & x2'==0 & x3'==0 & x4'==0 & x5'==0 & x6'==0} goto P5; //T5 orphan
  when tmsg1 <= x5 <= tmsg2 sync a do {x1'==x1 & x2'==0 & x3'==0 & x4'==0 & x5'==0 & x6'==x6} goto
P2; //T6 renewing ttl

loc P5: while true wait {x1'==0 & x2'==0 & x3'==0 & x4'==0 & x5'==0 & x6'==0}; //orphan agent, not
desired state

initially: P1 & x1 == 0 & x2 == 0 & x3 == 0 & x4 == 0 & x5 == 0 & x6==0;
end
```

Figura 5.8. Código para la descripción del Autómata de entrada al PHAVer

A continuación se mostrarán algunas de las gráficas obtenidas usando el analizador PHAVer, donde se grafica el tiempo total en ejecución del agente ejecutor (parámetro *tmax* modelado con el cronómetro x_1) contra distintos parámetros como:

1. El cronómetro que controla el *ttl* asignado al agente (x_2).
2. El tiempo total real del agente en el sistema (cronómetro x_6), es decir, tiempo en ejecución + tiempo en estado de bloqueo por renovación de ttl; para ver observar el tiempo real de ejecución del agente.
3. El tiempo en que llegan los mensajes de aprobación de ttl (cronómetro x_5).

En las Figuras 5.9, 5.10 y 5.11 se muestran algunos de los resultados obtenidos.

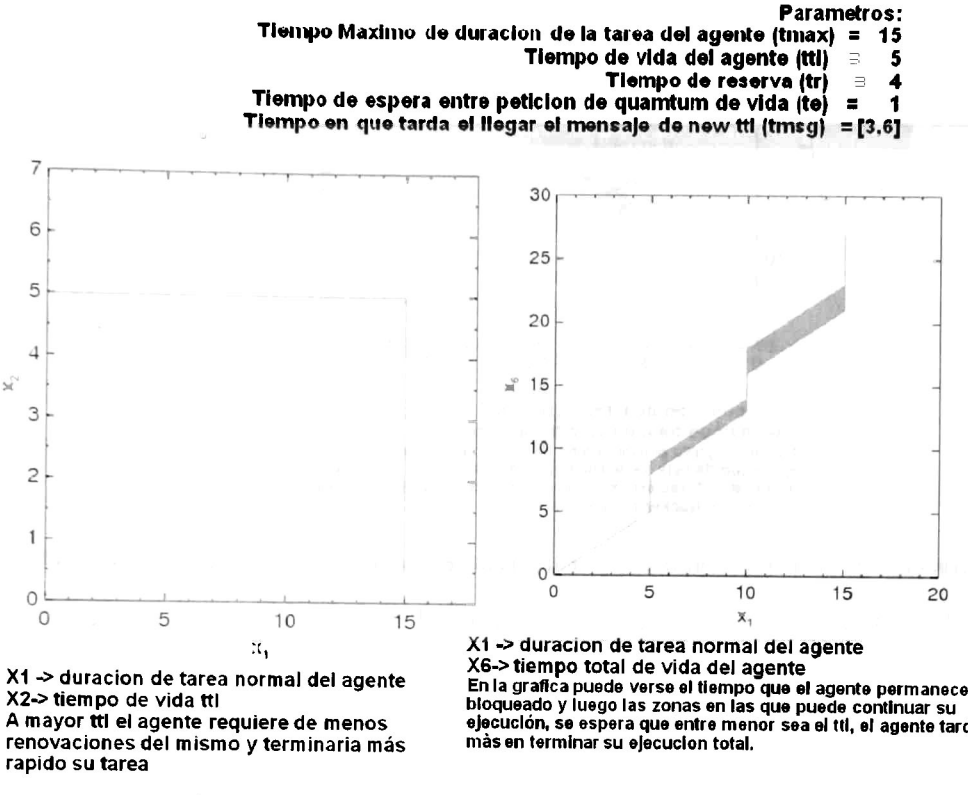


Figura 5.9. Gráficas de análisis temporal para el agente ejecutor de COPAM usando PHAVer

En la Figura 5.9 se muestran dos gráficas; en la izquierda se muestra el tiempo de ejecución del agente (x_1) contra el cronómetro que controla el tiempo de vida del agente (x_2); en esta gráfica puede observarse las veces que el cronómetro x_2 es reiniciado, es decir el número de renovaciones de *ttl* que tiene que realizar el agente de acuerdo a los parámetros definidos. En la parte derecha se muestra el tiempo total de vida del agente en el sistema, es decir, su tiempo de ejecución considerando el tiempo que permanece en estado de bloqueo mientras se renueva su tiempo de vida.

En la gráfica de la Figura 5.10 se disminuyó el parámetro de *ttl* definido en la Figura 5.9; aquí puede observarse, de acuerdo al nuevo parámetro de *ttl,* que el agente ejecutor pasaría más tiempo en ejecución en el sistema que con el valor definido en la Figura 5.9.

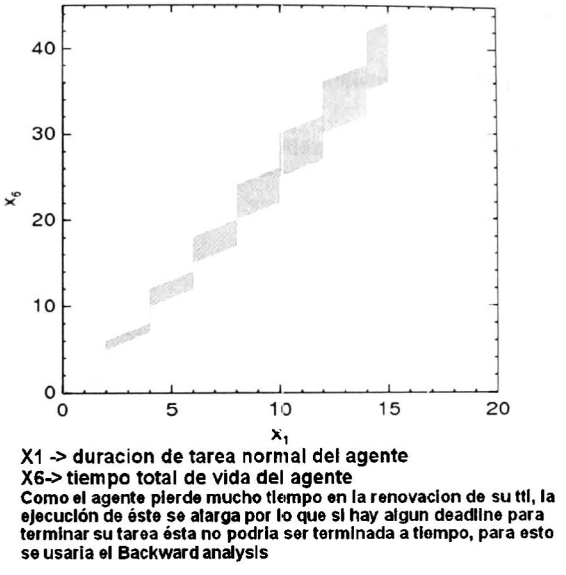


Figura 5.10. Gráfica análisis de duración total del agente ejecutor en el sistema, disminuyendo el *ttl*

En la Figura 5.11 se muestra la variación en el tiempo máximo en que el agente ejecutor puede permanecer en estado de bloqueo debido a la renovación de su *ttl*. En la gráfica de la izquierda con los parámetros definidos en la Figura 5.9 donde el tiempo de arribo del mensaje de renovación de *ttl* es mayor al tiempo total (*tr*) que el agente ejecutor posee para renovar su tiempo de vida. La gráfica derecha se obtiene haciendo el tiempo de arribo de mensaje de renovación de *ttl* menor que *tr*.

El uso de PHAVer es de gran ayuda para observar las regiones de alcanzabilidad de los cronómetros que se utilizan, por ejemplo, en el modelado de un protocolo de control de agentes. Sin embargo, ya que PHAVer solo trabaja con autómatas con cronómetros, tiene que convertirse el modelo en Red de Petri a un autómata equivalente, lo cual no siempre puede lograrse cuando la PN no es acotada. Por lo que resulta altamente conveniente evitar el paso por la obtención del autómata equivalente. En la siguiente sección se mostrará cómo se puede realizar un análisis temporizado simulando directamente los modelos en GSWPN.

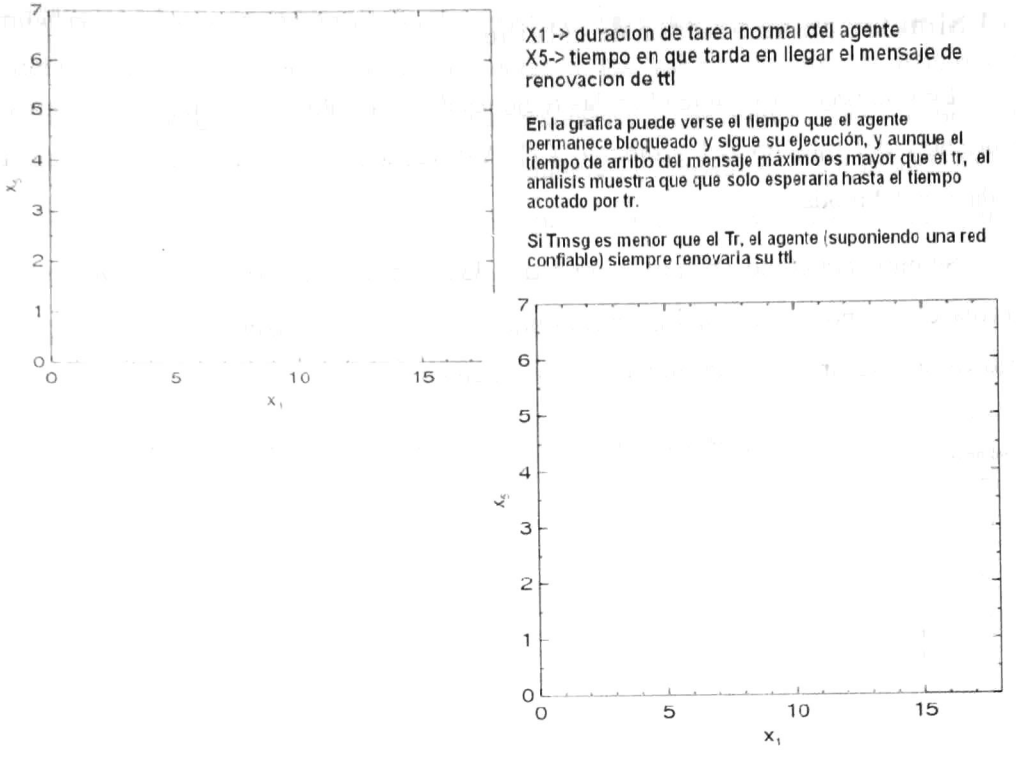


Figura 5.11. Análisis del SWA, variaciones en el tiempo de llegada de aprobación de *ttl*

5.3 Simulación de los modelos en GSWPN

La simulación directa de un modelo en GSWPN sin traducirlo a un autómata temporizado, agiliza el análisis del comportamiento del sistema modelado de acuerdo a ciertos parámetros definidos. En el caso de modelado de protocolos de control de agentes, se obtienen las regiones de alcanzabilidad de los cronómetros que controlan el comportamiento del agente, por ejemplo puede obtenerse el tiempo total que el agente pasaría en el sistema considerando el tiempo que permanece en estado de bloqueo y así determinar si dicho agente puede cumplir con un plazo de terminación determinado, etc.

5.3.1 Simulación de un caso de estudio

En esta sección se obtendrán las regiones de alcanzabilidad de los cronómetros más importantes del modelo del protocolo COPAM reducido en la Figura 5.6 mediante la simulación del modelo.

Se mostrará cómo obtener las regiones de alcanzabilidad para el cronómetro x_2 (que controla el tiempo de vida del agente móvil) del modelo de la Figura 5.6, para observar cómo varía el valor de este cronómetro con el tiempo.

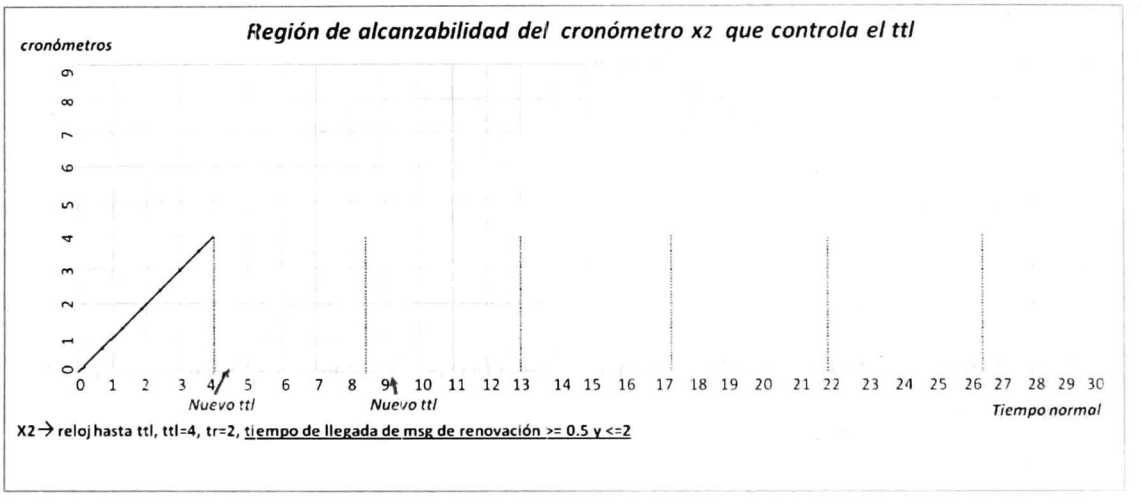


Figura 5.12. Región de alcanzabilidad del cronómetro x_2 tomando el valor mínimo del intervalo de llegada de mensaje de renovación de *ttl*

Para el modelo de la Figura 5.6, debido a que se tiene una transición (t6) que se dispara cuando se cumple un intervalo de tiempo *(tmsg1<=x5<=tmsg2)*, significa que la transición puede dispararse en cualquier momento dentro del intervalo induciendo así una región de posibles valores para los cronómetros. En la Figura 5.12 se muestra la variación en el tiempo del cronómetro x_2 si se toma el valor mínimo del intervalo (x5==tmsg1) inducido por t6. Como puede observarse, esta información no está completa debido a que no se han considerado todos los casos inducidos por el intervalo. Si se considera el valor máximo del intervalo (x5==tmsg2), se obtiene la gráfica de la Figura 5.13.

Si se agrega a la misma gráfica el resultado de la simulación tomando el valor mínimo y el resultado del a simulación tomando el valor máximo del intervalo en t_6, se obtienen zonas o regiones de alcanzabilidad del cronómetro x2 inducidas por las regiones

delimitadas por el valor máximo y mínimo, como se observa en la Figura 5.14. Cada región está delimitada por las líneas correspondientes a los valores mínimo y máximo del intervalo. Como puede observarse en la parte superior de la Figura 5.14, se forman 4 regiones y a partir de la región 4 las regiones empiezan a empalmarse. En la parte inferior de la Figura 5.14 se muestra cada región representando los posibles valores para el cronómetro x_2 según los valores de los parámetros para el modelo.

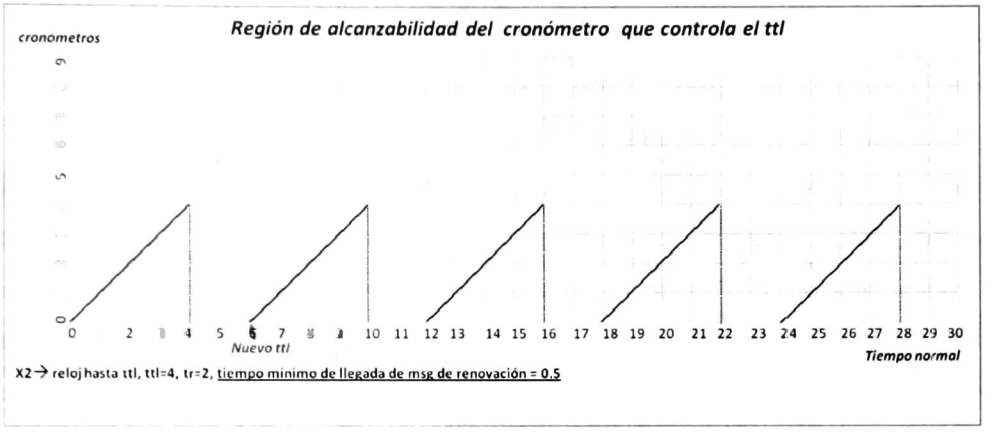


Figura 5.13. Región de alcanzabilidad del cronómetro x_2 tomando el valor máximo del intervalo de llegada de mensaje de renovación de *ttl*

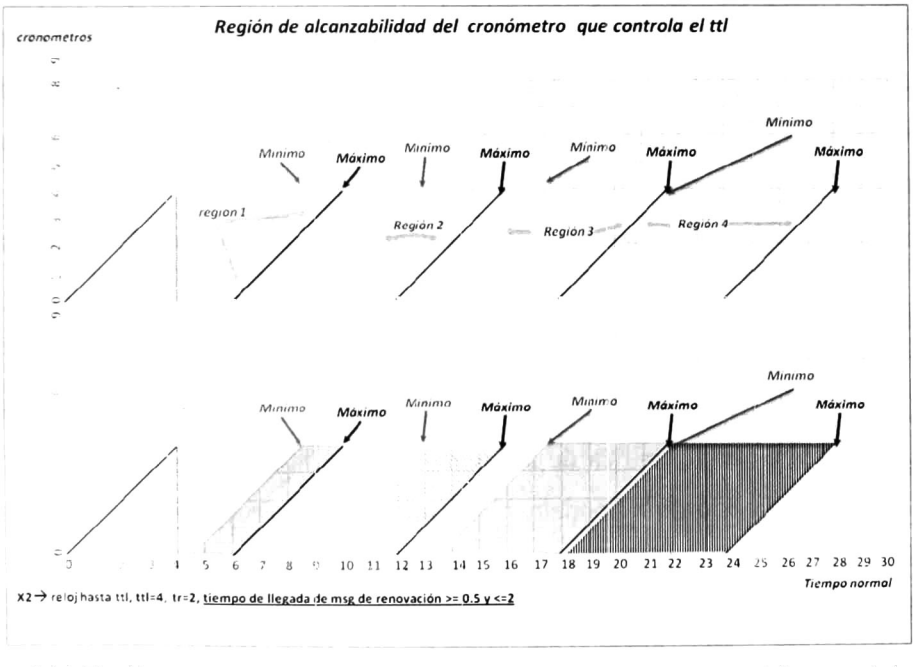


Figura 5.14. Región de alcanzabilidad del cronómetro x_2 tomando el valor mínimo y máximo del intervalo de llegada de mensaje de renovación de *ttl*

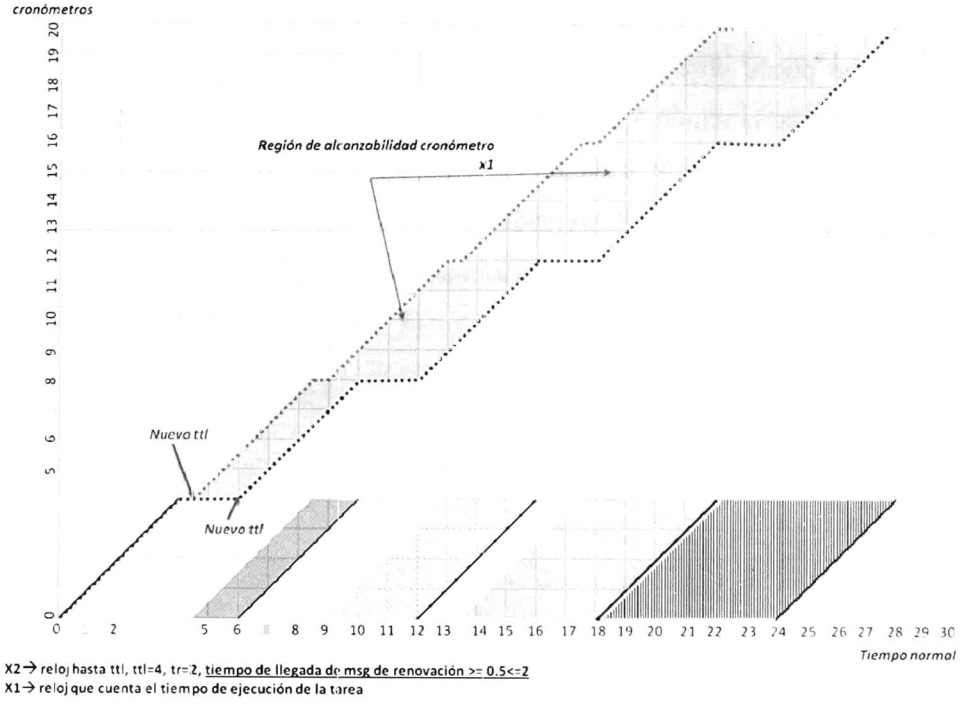


Figura 5.15. Regiones de alcanzabilidad para los cronómetros x_1 y x_2

Ahora bien, si se agrega el cronómetro x_1 a la gráfica de la Figura 5.14 se obtendría la región en gris de la Figura 5.15, la cual indica los estados alcanzables para el cronómetro x_1 de acuerdo a los parámetros dados; así puede observarse a determinado tiempo de ejecución de la tarea del agente (x_1) en qué instante realmente se está ejecutando según el tiempo normal o tiempo de simulación. Como ya se mencionó, se forma una región de valores debido al intervalo de tiempo en la transición t_6 del modelo en GSWPN en la Figura 5.6.

Suponiendo que el agente ejecutor termina su tarea (modelada por x1) en el intervalo expresado por 18<=x1<=20. Si se agrega esta información a la gráfica de la Figura 5.15 se obtendría la gráfica de la Figura 5.16, en la que se determinan los estados aceptables (o de aceptación) del modelo. Con esto puede observarse el tiempo real (tiempo de ejecución de la tarea más el tiempo que el agente permanece en bloqueo por la renovación de tu tiempo de vida) que le toma al agente ejecutor realizar una tarea que termina entre 18 y 20 unidades de tiempo.

Como puede observarse en la Figura 5.16, al agente ejecutor le tomaría entre 20 y 28 unidades de tiempo terminar la tarea modelada por el cronómetro x1 de acuerdo a los parámetros dados para el *ttl*, *tr* y el intervalo de llegada del mensaje de renovación de *ttl*. Si el agente ejecutor tuviera que cumplir un plazo determinado (línea amarilla en la Figura 5.16), es decir, terminar su tarea dentro de un tiempo determinado, con la región de alcanzabilidad del cronómetro x1 podría observarse si el agente puede cumplir ese plazo o no.

Si se varían los valores de *ttl* y *tr* al modelo (ttl=3, tr=4) y se tuvieran los mismos valores de terminación de la tarea ([18,20]) del agente móvil y el plazo a cumplir (deadline=28), se obtendría la gráfica de la Figura 5.17. En esta figura puede observarse que el agente puede o no cumplir con el plazo determinado, es decir, el plazo está dentro del intervalo de terminación real del agente, pero hay muchos estados del agente móvil que no caen dentro del intervalo o plazo deseado (línea amarilla), por lo que la probabilidad de que el agente termine fuera del plazo determinado es muy alta. Una corrección en los parámetros pudiera corregir ese problema.

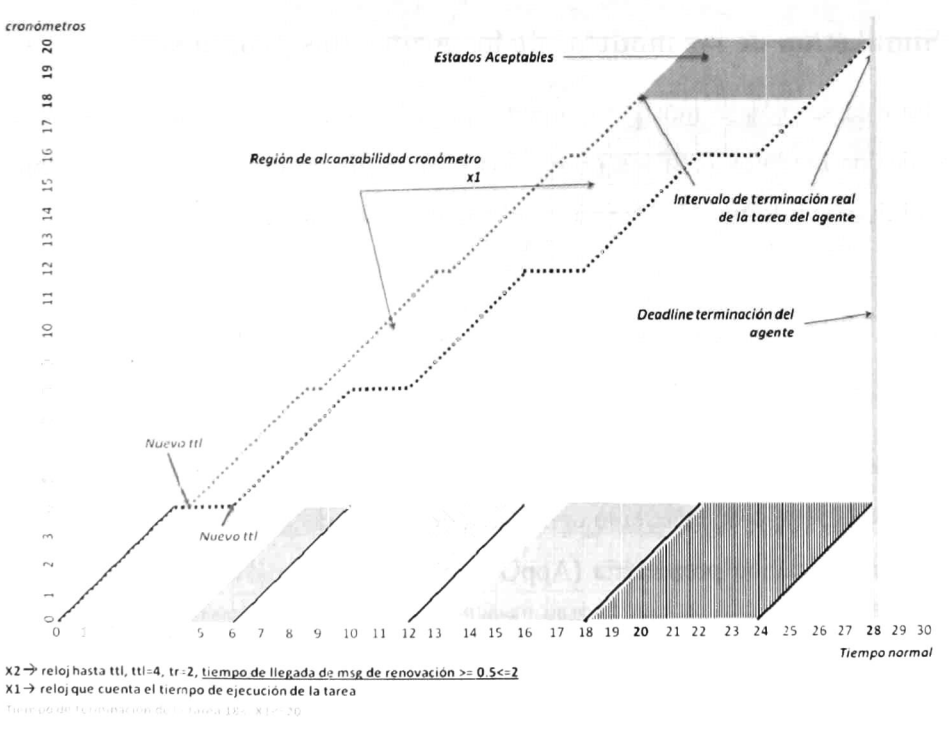


Figura 5.16. Regiones de alcanzabilidad para los cronómetros x_1 y x_2, dando un valor de terminación de la tarea modelada y un deadline

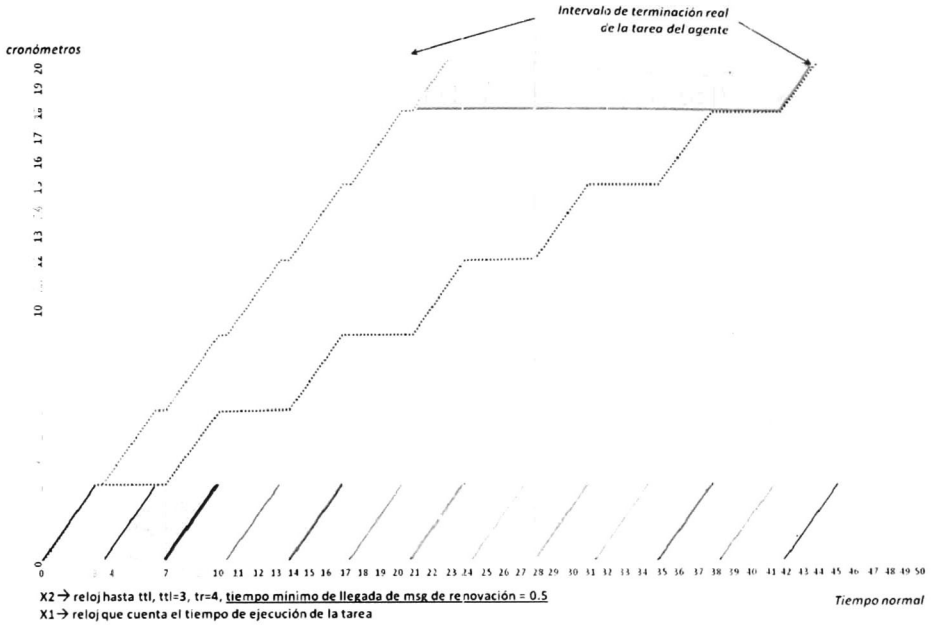


Figura 5.17. Regiones de alcanzabilidad para los cronómetros x_1 y x_2, donde muchos estados de terminación del agente quedan fuera de un deadline.

5.3.2 Simulación de los modelos de los protocolos propuestos

En esta sección se mostrarán algunas gráficas obtenidas mediante la simulación de algunos de los modelos para los protocolos propuestos en esta tesis, con el objetivo de mostrar el tipo de información que puede obtenerse de los modelos.

5.3.2.1 Simulación de los modelos para el protocolo de localización de agentes móviles

En la Figura 5.18 se muestran las regiones de alcanzabilidad para los cronómetros que modelan el tiempo de ejecución de la tarea de localización de agentes móviles de: 1) en color rojo, la aplicación propietaria (AppOwner) del agente móvil modelada en la Figura 4.13; y, 2)en azul, el modelo del Agente Administrador (AM) modelado en la Figura 4.14 cuando se trata de una localización local. Los parámetros o valores máximos dados a cada agente como límite para la terminación de esta tarea se fijaron en tmax=15 para ambas redes.

Al incluir en la Figura 5.18 los instantes en que pueden ocurrir los disparos importantes de las transiciones (transiciones que afectan el valor de los cronómetros x1), puede determinarse en qué instante pueden comenzar ($[t_i$) y en qué instante máximo pueden ser disparadas ($t_i]$) dichas transiciones. También puede determinarse, con los parámetros dados, cual es el tiempo en total maximo que llevaria realizar esta tarea.

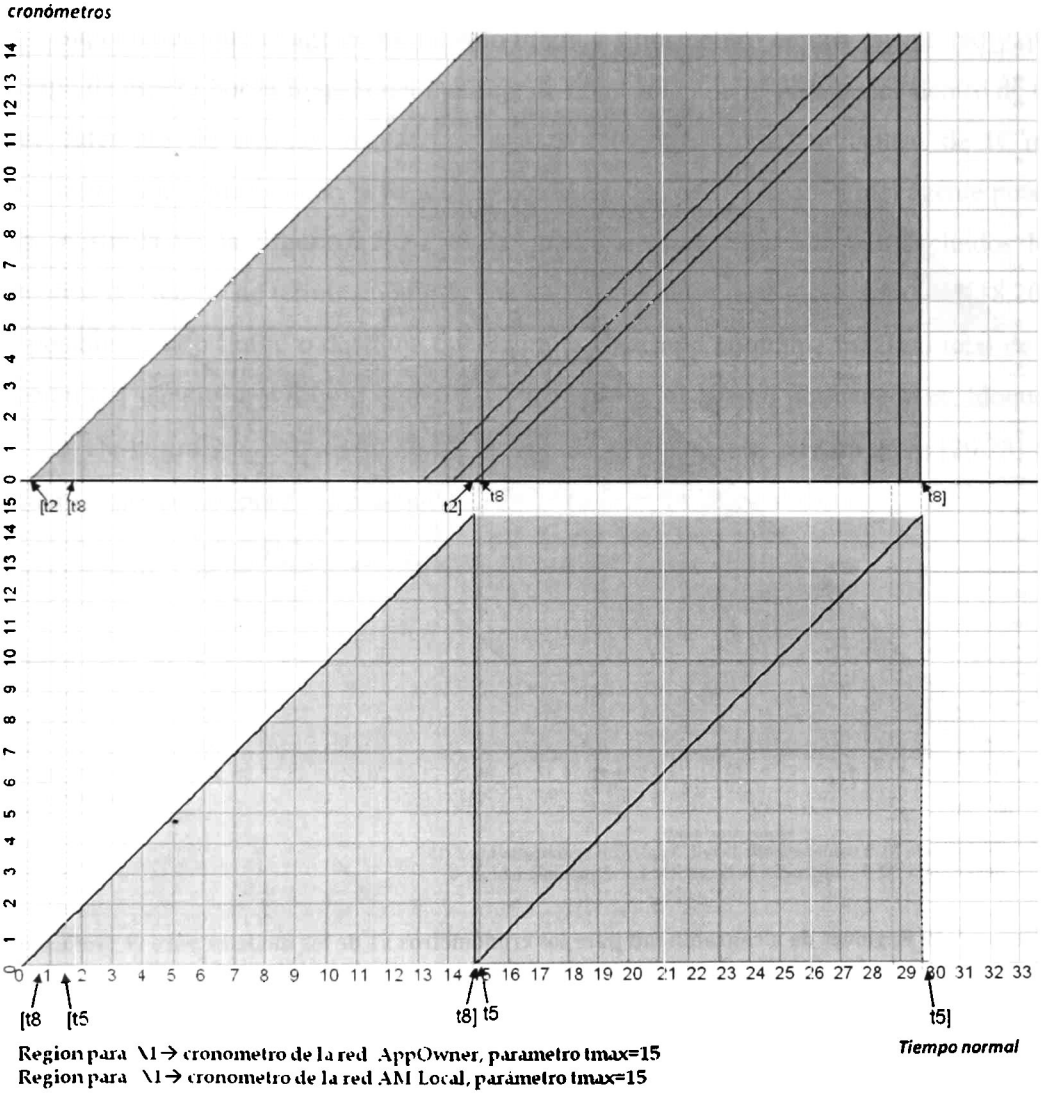


Figura 5.18 Regiones de alcanzabilidad para los cronómetros x1 de los modelos para el protocolo de localización de agentes móviles

Si se disminuye el tiempo máximo de ejecución de la tarea de localización del agente móvil del modelo para el AM local de la Figura 4.14, tmax=8; se obtendrían las gráficas mostradas en la Figura 5.19.

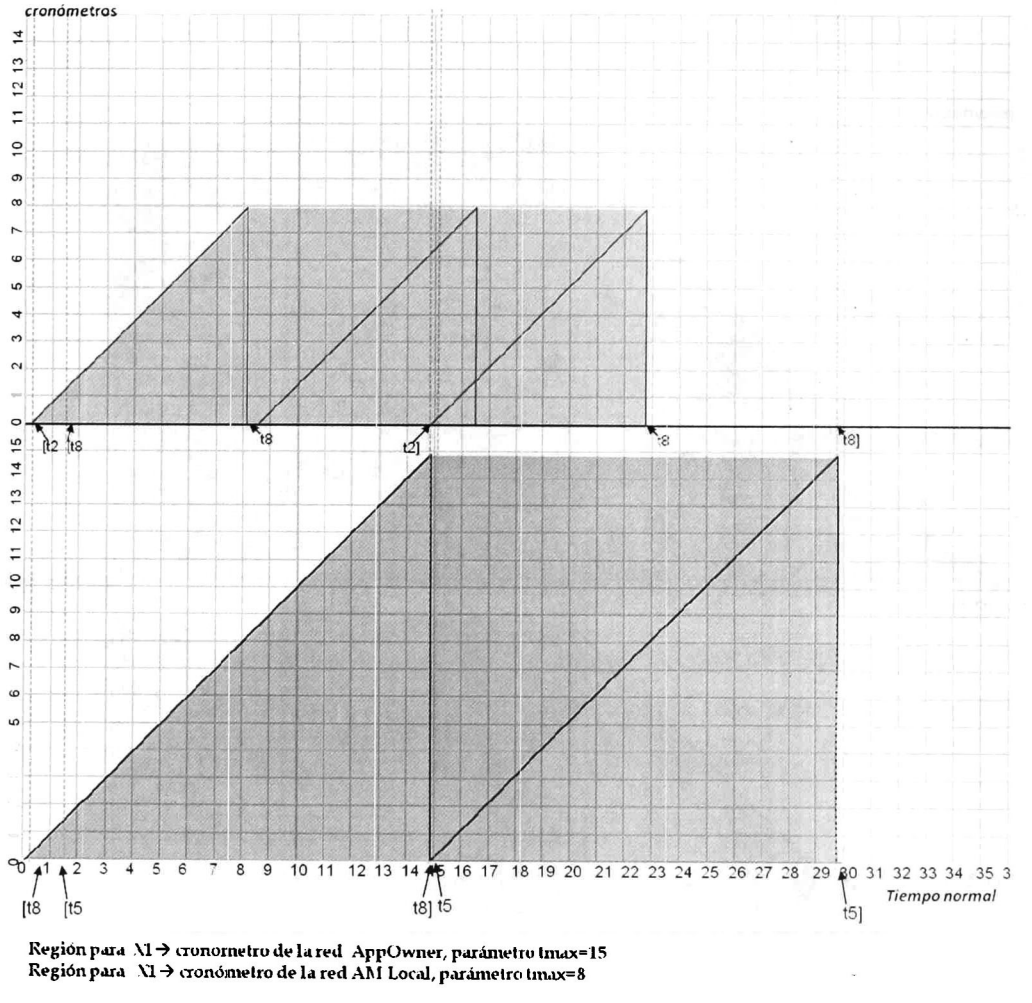


Figura 5.19 Regiones de alcanzabilidad para los cronómetros x1 de los modelos para el protocolo de localización de agentes móviles, disminuyendo el parámetro tmax.

5.3.2.2 Simulación de los modelos para el proceso de renovación de tiempo de vida de un agente móvil

En esta sección se mostrará la región de alcanzabilidad de la ejecución del Agente móvil (cronómetro x_1) durante el proceso de renovación de tiempo de vida (*ttl*) modelado en la Figura 4.19.

Suponiendo que el agente móvil solo migra a sitios dentro de una misma red y que el tiempo de espera por la llegada del mensaje de renovación de *ttl* permanece dentro de un mismo intervalo de tiempo mientras el agente móvil se encuentre dentro de la red (0.5<=tmsg<=2), la gráfica con la región de alcanzabilidad de la ejecución del agente puede ser la mostrada en la Figura 5.20. Para la gráfica de esta figura se han incluidos los parámetros de tiempo de terminación máxima de la tarea normal del agente móvil ([18,20]) y un posible tiempo límite o deadline que delimita el tiempo aceptable máximo total de la ejecución del agente móvil considerando el tiempo que el agente permanece en bloqueo ( ). De la gráfica de la Figura 5.20, se obtiene que el agente tardara entre [20,28] en terminar su ejecución, por lo que cumpliría con el tiempo límite establecido.

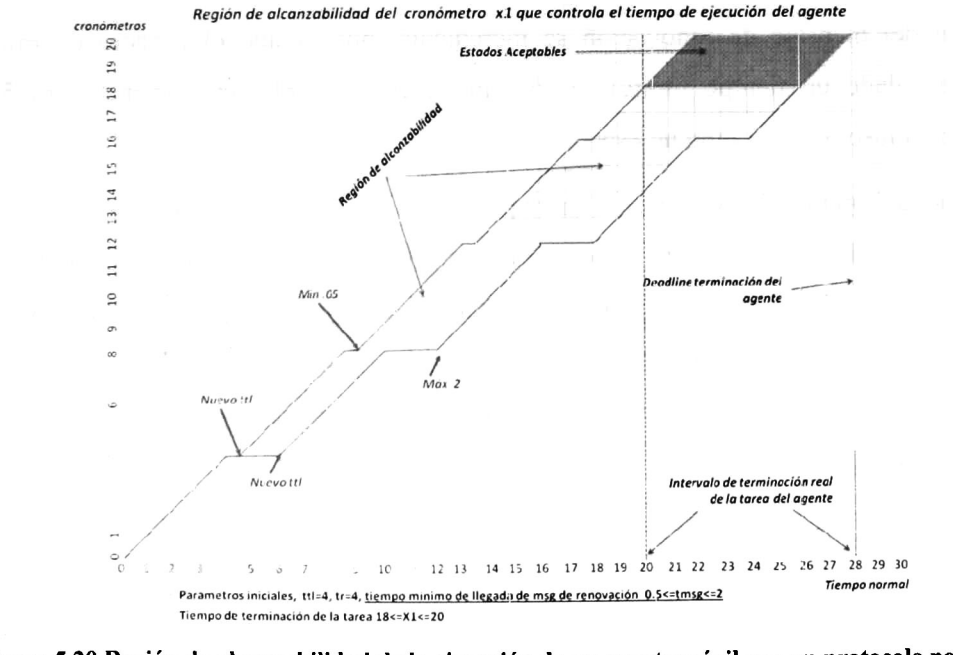


Figura 5.20 Región de alcanzabilidad de la ejecución de un agente móvil con un protocolo normal de renovación de tiempo de vida

En un Flujo de trabajo interorganizacional, el agente móvil tiene que viajar a las redes de otras compañías para realizar operaciones referentes al caso que maneja. Debido a que la comunicación se realiza por medio de internet, el tiempo de espera del mensaje de renovación de *ttl* se incrementaría cuando el agente móvil dejara la red donde se encuentra su aplicación propietaria y por tanto el agente manejador que aprueba la renovación de su *ttl.*

Suponiendo que se tiene un agente móvil en el que, inicialmente, el mensaje de renovación de *ttl* tarda entre 0.5 y 2 unidades de tiempo (0.5<=tmsg<=2); después dicho agente migra a otra red incrementando el tiempo de la llegada del mensaje a un intervalo entre 1 y 3 unidades de tiempo (1<=tmsg<=3), y si no se utilizara el proceso de renovación de *ttl* propuesto en esta tesis, se tendría la región de alcanzabilidad mostrada en la Figura 5.21. Como puede observarse en la figura, al incrementar el tiempo en el que el agente móvil permanece en estado de bloqueado por la espera del mensaje de renovación de *ttl*, se incrementa el tiempo total que el agente permanece en el sistema ([21,29]), por lo que la región de estados aceptables del agente móvil se decrementa. El tiempo que el agente permanece en el sistema se incrementa cuando el agente continúa migrando a redes donde el retraso del mensaje de renovación se incrementa, por lo que el espacio de estados aceptables (dado un tiempo límite), se disminuye considerablemente (ver Figura 5.22) pudiendo incluso quedar como un espacio vacío.

En la Figura 5.22 se muestra la región de alcabilidad de un agente móvil cuyo tiempo de llegada de mensaje de renovación de *ttl* ha llegado a [3,5] unidades de tiempo, sin embargo, debido a que el parámetro de tiempo de reserva (*tr*) del agente móvil es *tr=4*, el agente móvil solo espera hasta 4 para renovar su *ttl*, por lo tanto entre mayor sea el tiempo máximo de llegada de mensaje de renovación, el agente solo esperará máximo *tr* unidades de tiempo para renovar su *ttl*.

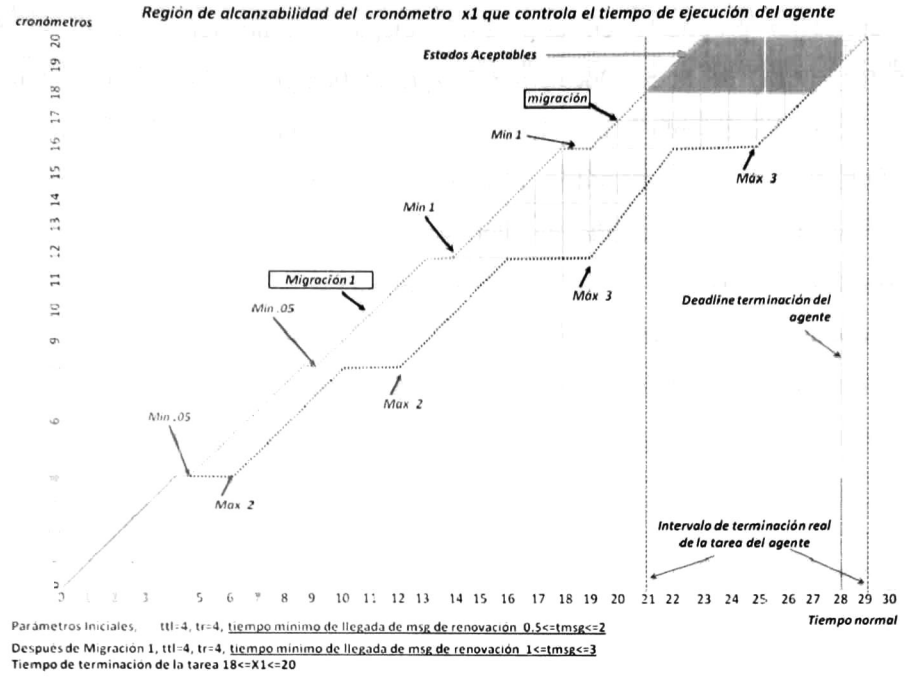


Figura 5.21 Región de alcanzabilidad de la ejecución de un agente móvil con un protocolo normal de tiempo de vida después de migrar a una red diferente

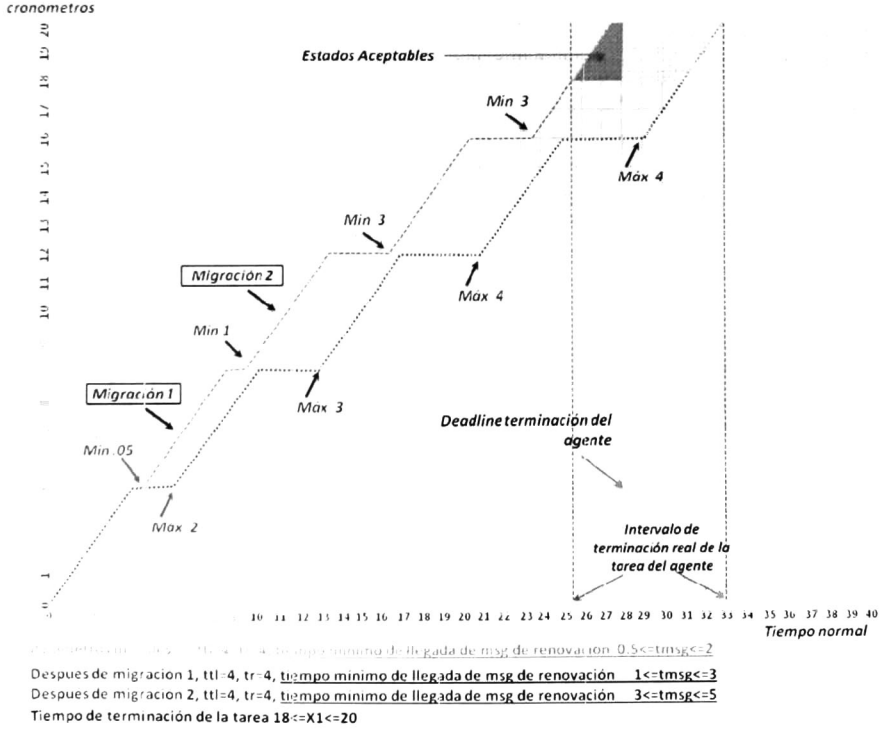


Figura 5.22 Región de alcanzabilidad de la ejecución de un agente móvil con un protocolo normal de tiempo de vida después dos migraciones a redes distintas

El protocolo propuesto en esta tesis delega la renovación de *ttl* al Agente Administrador (AM) de la red local, por lo que el tiempo de llegada del mensaje de renovación no se incrementa como cuando el agente envía el mensaje de renovación a través de internet. Por lo tanto, podría decirse que para el Agente móvil permanece más tiempo en ejecución que en estado de bloqueo al utilizar el protocolo propuesto. Una comparación del tiempo que el agente móvil consume en su ejecución utilizando el protocolo propuesto en esta tesis y el protocolo COPAM, puede verse en la Figura 5.23, donde se observa que, con el nuevo protocolo de renovación de *ttl* el agente móvil termina sus tareas en un tiempo menor al que se emplearía si se utilizara el protocolo COPAM.

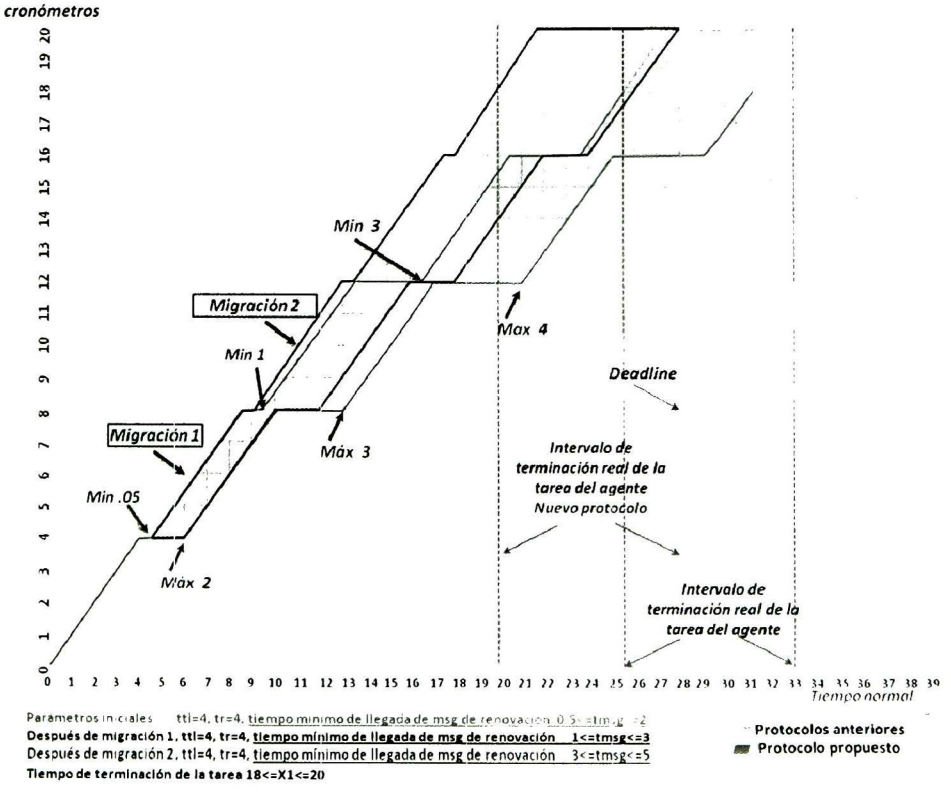


Figura 5.23 Comparación de las regiones de alcanzabilidad de dos Agentes móviles donde uno sigue el protocolo COPAM y el segundo el protocolo propuesto en esta tesis

Con la ayuda de estas gráficas se ha ilustrado la utilidad del análisis basado en la ejecución de modelos GSWPN. Se ha mostrado un análisis por regiones de alcanzabilidad de algunos de los cronómetros utilizados en los modelos y se mostró un ejemplo del tipo de información que se podría obtener con la ayuda de estas gráficas, aunque, como pudo observarse, la información obtenida está en base a la interpretación que se le dé a cada cronómetro. Por lo tanto, una herramienta que permita realizar un análisis de este tipo sería de gran utilidad cuando se modela con GSWPN.

Conclusiones

En esta tesis se ha abordado el problema de la tolerancia a fallas en los sistemas automatizados de flujo de trabajo basados en agentes móviles, en un ambiente inter-organizacional. Las aportaciones de este trabajo pueden resumirse en dos resultados. El primero se relaciona a la propuesta de protocolos de dos tipos; en el primer tipo se definieron protocolos orientados al control de agentes móviles (incluyendo tareas de localización, terminación y control de agentes huérfanos) y en el segundo tipo se propuso un protocolo orientado a la recuperación de agentes móviles. Ambos tipos de protocolos se basaron en la división del problema en diferentes redes, una para cada compañía o socio de negocio en un Flujo de Trabajo Interorganizacional. Además, en cada red se incorporó el uso de un Agente Administrador para controlar a los agentes móviles que se ejecutaban dentro de la red a la que pertenece cada Agente Administrador. Así, en lugar de poseer a una entidad que controle a todos los agentes del sistema (como lo realizan propuestas similares de otros autores), se tienen varias entidades que realizan el control por zonas o redes lo cual permite un mejor manejo de los agentes.

En los protocolos de control de agentes móviles, aunque se adaptan algunos conceptos utilizados en otras propuestas (camino de proxies y tiempo de vida), al emplear varias entidades encargadas del control de los agentes se logra un control diferente y con muchas ventajas. Algunas de las mejoras que se tienen al utilizar este tipo de distribuciones se obtienen principalmente al reducir el camino de proxies para localizar a un agente móvil, lo cual también permite una terminación activa más rápida. Además, al delegar el proceso de renovación de tiempo de vida de un agente móvil al Agente Administrador de la red actual, se reduce el tiempo en que el agente móvil permanece en estado de bloqueo utilizado en este proceso y por lo tanto se reduce el tiempo total que agente utiliza en el sistema para realizar sus tareas.

En el protocolo de recuperación de agentes se reutilizan los conceptos de agentes testigo, *checkpoints* y *logs*; sin embargo se propuso una técnica para disminuir la cantidad de recursos utilizados limitando el número de testigos y reduciendo el número de mensajes que se intercambian en la etapa de monitoreo. Además, a diferencia de otras propuestas de

recuperación de agentes, cuando se presenta la falla del agente móvil y de todos los testigos (caso en el que las propuestas anteriores no detectan la falla del agente), aún puede detectarse una falla en el agente móvil y por lo tanto recuperar su estado, ya que se cuenta con un Agente Administrador que controla el tiempo de vida en el sistema (utilizado en el protocolo de control de agentes). En consecuencia, se asegura que a pesar de la falla de todos los agentes testigo para el protocolo de recuperación de agentes móviles, al presentarse la falla en la ejecución del agente móvil, ésta eventualmente será detectada por el sistema.

En base a lo anterior, se ha observado que la estrategia propuesta puede funcionar como la base de un algoritmo más complejo, en el cual se permita delegar algunas de las funciones de los administradores del sistema a otros nodos, con el objetivo de distribuir la carga de trabajo a través de dicho sistema. Además, debido a la manera como se planteó el modelado del sistema se pueden incorporar otras características tales como la interoperabilidad de las diferentes bases de datos existentes en el sistema. Todo lo anterior es posible gracias a la estructuración del algoritmo propuesto.

El segundo grupo de resultados está relacionado al modelado y análisis de protocolos de control de Agentes móviles. Se propuso una extensión a las Redes de Petri Temporizadas para el modelado de los protocolos propuestos en esta tesis, la cual también puede utilizarse para el modelado de cualquier sistema que requiera modelar tareas que puedan ser suspendidas y que involucren el tiempo como factor del sistema. Con este tipo de Redes de Petri (GSWPN) se tiene la ventaja de que se pueden modelar tareas interrumpibles, es decir, tareas que pueden detenerse por cierto tiempo y después reanudarse en el estado que tenían antes de ser interrumpidas. Además de que se propone una base formal para el modelado de protocolos de control de agentes, se logra una mejor definición de los mismos y se puede realizar un análisis temporizado mediante la simulación de los modelos o mediante el uso de herramientas de análisis existentes (PHAVer).

Al permitir el uso de cronómetros globales, las GSWPN permiten modelar los diferentes estados de una tarea que puede ser interrumpida en cualquier momento, lo cual no es soportado por anteriores propuestas, ya que todas manejan cronómetros asociados a

una sola transición; incluso, los modelos que pueden construirse con la extensión propuesta capturan comportamientos más complejos, por lo que las definiciones previas (Allahham & Alla, 2007; Berthomieu et al., 2007) resultan subclases de este nuevo formalismo.

El trabajo aquí presentado constituye un paso adelante en el diseño de sistemas confiables para la automatización del flujo de trabajo; sin embargo podemos señalar desde ahora algunas extensiones posibles, las cuales pueden abordarse a corto y mediano plazo.

- Implementación y pruebas de la propuesta de los protocolos utilizando JADE como plataforma de desarrollo de agentes.
- Mejoras al protocolo de recuperación definiendo mecanismos similares al *rollback,* los cuales permitan deshacer las operaciones que un agente móvil ha realizado en un sitio, ya sea cuando se trate de un sitio que se recuperó de una falla, o cuando se desee eliminar activamente a un agente móvil, para dejar al sistema en un estado equivalente al que tenía antes de que el agente móvil se ejecutara.
- Incluir mecanismos para la distribución de carga tendientes a disminuir el cuello de botella generado por la administración en cada red en un solo punto de entrada (*nodo gateway*).
- Desarrollo de una herramienta que permita la edición de modelos siguiendo el formalismo de las GSWPN proporcionando las funcionalidades de: 1) simulación de los modelos y 2) análisis temporizado automático de los diferentes cronómetros de los modelos mediante el uso de gráficas con los estados alcanzables de los cronómetros.

REFERENCIAS

Adam, N.R.; Atluri, V. & Huang, W.K. (1998). Modeling and Analysis of Workflows Using Petri Nets. *Journal of Intelligent Information Systems.* Vol. 10, Issue 2, March 1998, pp. 131-158, ISSN : 0925-9902.

Ahn, J. & Min, S.G. (2005). Fault-Tolerant and Scalable Protocols for Replicated Services in Mobile Agent Systems. *Lecture Notes in Computer Science,* 2005, ISSU 3516, pp. 679-686, ISSN: 0302-9743, Springer-Verlag, Germany, 2005.

Allahham, A. & Alla, H. (2007). Post and Pre-initialized Stopwatch Petri Nets, *In Proceedings of the 1st IFAC Workshop on Dependable Control of Discrete Systems (DCDS'07),* France, 2007

Awerbuch, B. & Peleg, D. (1995). Concurrent online tracking of mobile users, Journal *of the ACM , Vol.* 42, No. 5, 1995. pp. 221 - 233

Bagnara, B. ; Hill, P.M. ; Ricci, E. & Zaffanella, E. (2002). *The Parma Polyhedra Library User's Manual.* Department of Mathematics, University of Parma, Parma, Italy, release 0.4, edition July 2002. Available at http://www.cs.unipr.it/ppl/.

Baumann, J. (1997). A protocol for orphan detection and termination in mobile agent systems, Technical Report Nr. 1997/09, Faculty of Computer Science, University of Stuttgart, Germany. 1997.

Baumann, J. (1999). Control Algorithms for Mobile Agents, tech. report of Stuttgart University,1999, pp.59-79

Bellifemine, F.; Caire, G.; Trucco, T. & Rimassa, G. (2007). *Jade programmer's guide,* JADE 3.5. 2007.

Berthomieu, B.; Lime, D. & Roux, O. (2007). Reachability Problems and Abstract State Spaces for Time Petri Nets with Stopwatches. *Journal of Discrete Event Dynamic Systems Theory and Applications (jDEDS),* 17(2):133-158, DOI 10.1007/s10626-006-0011-y, Springer Science, April 2007.

Black, Q.; Hutchinson, N.; Jul, E.; Levy, H. & Carter, L. (1986). Distribution and abstract types in Emerald, *IEEE Transactions on Software Engineering,* Vol. 13, No. 1, 1986. pp. 65 - 76.

Cabri, G.; Leonardi, L. & Zambonelli, F. (1998), Mobile Agent Technology: Current Trends and Perspectives, *Congresso annuale AICA'98,* Napoli, Italy, November 1998.

Cassez, F. & Roux, O. (2006). Structural Translation from Time Petri Nets to Timed Automata, *Journal of Systems and Software,* 79(10):1456-1468, 2006

Eshuis, R. & Dehnert, J. (2003). Reactive Petri Nets for Workflow Modeling. In *Application and Theory of Petri Nets 2003,* vol. 2679, of Lecture Notes in Computer Science, pp. 296-315, ISBN: 3-540-40334-5, Springer Verlag, Berlin.

Flores Badillo, M. (2006). Coordinación de Flujo de Trabajo basado en Sistemas de Agentes Móviles, *Tesis para obtener el grado de Maestro en Ciencias en el CINVESTAV,* agosto 2006.

Flores-Badillo, M & López-Mellado, E. (2008). Mobile Agent based Automation of Distributed Workflow Processes, in *Proceedings of the IEEE International Conference on System of Systems Engineering, (SOSE'08)* June 2nd - 4th, 2008, Portola Plaza Hotel, Monterey, California, USA. ISBN: 978-1-4244-2172-5, pp. 1-6

Flores-Badillo, M. & López-Mellado, E. (2010). Multilevel Specification and Development of Mobile Agent Software for Workflow Processes Automation, capítulo del libro *"Petri Nets: Applications",* publicado por IN-TECH, Febrero 2010, ISBN: 978-953-307-047-6.

Flores-Badillo, M.; López-Mellado, E. & Padilla-Duarte, M. (2009). Modelling and Simulation of Complex Workflow Processes using Multi-level Petri Nets, *International Journal of Simulation and Process Modelling (IJSPM),* Vol. 4, No. 3/4, pp. 205-214, 2009. ISSN: 1740-2123.

Fowler, R.J. (1985). Decentralized object finding using forwarding addresses, *Ph.D. Thesis, Technical Report 85-12-1,* Department of Computer Science, University of Washington, USA. 1985.

Frehse G. (2006). *Language Overview for PHAVer, version 0.35, June 2006,* url: http://www-verimag.imag.fr/~frehse/phaver_web/index.html

Hevia, A. &Vasa, A. (2000). Fault tolerant protocols for mobile agents: A survey. https://wwwcse.ucsd.edu/classes/sp00/cse221/reports/hevvas.

Hollingsworth, D. (1995). The Workflow Reference Model, *The Workflow Management Coalition, Document Number WFMC-TC-1003*, sitio http://www.wfmc.org

Jochum, E. (1997). Konzeption und Implementierung eines Orphan-Detection Mechanismus nach dem Energiekonzept, *Student Thesis Nr. 1642*, Faculty of Computer Science, University of Stuttgart. 1997.

Jul, E.; Levy, H.; Hutchinson, N. & Black, A. (1988). Fine-grained mobility in the Emerald system, *in ACM Transactions on Computer Systems*, Vol. 6, No. 1, 1988. pp. 109 - 133.

Kummer, O. ; Wienberg, F. & Duvigneau M. (2002) *Renew – User Guide*, University of Hamburg, Department for Informatics, Theoretical Foundations Group.

Li, J.Q.; Fan, Y.S. & Zhou, M.C. (2004). Performance Modeling and Analysis of Workflow. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, Vol. 34, No. 2, 2004, pp 229-242, ISSN: 1083-4427.

Lime, D. & Roux, O. (2003). Expressiveness and Analysis of Scheduling Extended Time Petri Nets, *in 5th IFAC International Conference on Fieldbus Systems and their Applications, (FET 2003)*, Elsevier Science, July 2003.

Lu, R. & Sadiq, S. (2007). A Survey of Comparative Business Process Modeling Approaches. In *Proceedings of 10th International Conference on Business Information Systems BIS 2007*, LNCS, pp. 82-94, ISBN: 978-3-540-72034-8, Poznan, Poland, Springer-Verlag.

Lyu, M.R. & Wong, T.Y. (2003). A Progressive Fault Tolerant Mechanism in Mobile Agent Systems, in *Proceedings 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI2003)*, Orlando , Florida, July 27-30 2003, Volume IX, pp. 299-306.

Magnin, M.; Molinaro, P. & Roux, O. (2005). Improved Algorithm for Computing Exact State Space of Petri Nets with Stopwatches. *IRCCyN Technical report number R2005_15*, 2005.

Magnin, M.; Molinaro, P. & Roux, O. (2006)a. Decidability, Expressivity and State-Space Computation of Stopwatch Petri Nets with Discrete-time Semantics, *in Proceedings of the 8th International Workshop on Discrete Event Systems*, Ann Arbor, Michigan, USA, July 2006.

Magnin, M.; Molinaro, P. & Roux, O. (2006)b. How to deal *efficiently with Petri nets with stopwatches in discrete-time? IRCCyN Technical report number R2006_1*, 2006.

Magnin, M.; Molinaro, P. & Roux, O. (2006)c. An Efficient Method for Computing Exact State Space of Petri Nets with Stopwatches. *Electronic Notes in Theoretical Computer Science*, 144(3) : 59-77, Elsevier, 2006.

Marin, C.A. & Brena, R. (2005). Multiagent Architecture for Decentralized Workflow Process Execution, Technical Report, *Center for Intelligent Systems*, Tecnológico de Monterrey, marzo 2005

Marin, O.; Sens, P. ; Briot, J.P. & Guessoum, Z. (2001). Towards adaptive fault-tolerance for distributed multi-agent systems, In *Proc. of ERSADS'2001*, Bertinoro, Italy, 2001, pp. 195-201.

Nwana, H. (1996). Software Agents: an Overview, *Knowledge Engineering Review*, Vol. 11, No 3, pp. 205–244.

Padilla Duarte, A. (2007), A Mobile Agent Population Control Protocol (in Spanish), *MSc. Thesis, Cinvestav* Guadalajara Mexico. December 2007.

Padilla Duarte, M. (2008). Una Herramienta para la Generación de Software Distribuido Basado en Sistemas Multiagentes, *Tesis para obtener el grado de Maestría*, Cinvestav Unidad Guadalajara, 22 de Agosto 2008.

Pears, S.; Xu, J. & Boldyreff, C. (2003). Mobile Agent Fault Tolerance for Information Retrieval Applications: An Exception Handling Approach, *Proceedings of The Sixth International Symposium on Autonomous Decentralized Systems (ISADS'03)*, p.115, April 09-11, 2003.

Pleisch , S. & Schiper, A. (2000). Modeling Fault-Tolerant Mobile Agent Execution as a Sequence of Agreement Problems, in *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)*, p.11, ISBN:0-7695-0543-0, IEEE Computer Society,October 16-18, 2000.

Pleisch, S. & Schiper, A. (2001). FATOMAS - A Fault-Tolerant Mobile Agent System Based on the Agent-Dependent Approach. In *Proc. of the Conference on Dependable Systems and Networks*

(DSN). pp 215-224, ISBN: 0-7695-1101-5, IEEE. Computer Society, July 2001, Goteborg, Sweden.

PPL, The Parma Polyhedral Library, *url: http://www.cs.unipr.it/ppl/*

Repetto, M.; Paolucci, M. & Boccalatte, A. (2003). A design tool to Develop Agent-Based Workflow Management Systems, In *Proc. Italian Workshop, from Objects to Agents: Intelligent Systems and Pervasive Computing (WOA2003)*, pp. 100-107, ISBN: 88-371-1413-3, Villasimius, CA, Italy, September 2003. Pitagora Editrice Bologna.

Reijers, H.A. & van der Aalst, W.M.P. (2005).The Effectiveness of Workflow Management Systems: Predictions and Lessons Learned. *International Journal of Information Management*, Vol. 25, No. 5, pp. 458-472. ISSN: 0268-4012.

Rothermel, K. & Strasser, M. (1998). A Fault-Tolerant Protocol for Providing the Exactly-Once Property of Mobile Agents, *in Proc. of The 17th IEEE Symposium on Reliable Distributed Systems*, pp.100, 1998, ISBN: 0-8186-9218-9.

Sánchez-Herrera, R. (2004). Especificación Multinivel de Protocolos de Interacción en Sistemas de Agentes Móviles. *Tesis para obtener el grado de Maestría*, Cinvestav Unidad Guadalajara, 2004.

Savarimuthu, B.T.R & Purvis, M. (2004). A Collaborative Multi-Agent Based Workflow System, *Knowledge-Based Intelligent Information and Engineering Systems KES'2004*, Lecture Notes in Artificial Intelligence (LNAI), vol. 3214, pp. 1187-1193. ISSN: 0302-9743.

Savarimuthu, B.T.R.; Purvis, M. & Fleurke, M. (2004). Monitoring and Controlling of a Multi-agent based Workflow System, In *Proceedings of the second Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation*, Vol. 32, pp. 127-132, Dunedin, New Zealand, 2004, Australian Computer Society, Inc. Darlinghurst, Australia.

Shi, M.L.; Yang, G.; Xiang, Y. & Wu, S. (1998). Workflow Management Systems: A Survey. In Proc. IEEE International Conference on Communication Technology, ICCT'98, pp. 1-5, Beijing, China, 1998.

Silva, A. & Popescu-Zeletin, R. (1998). An Approach for Providing Mobile Agent Fault Tolerance. *In Proceedings of the 2d Int. Workshop on Mobile Agents*, pp. 14-25, Stuttgart, Germany, 1998

Spector, A. (1982). Performing Remote Operations Efficiently on a Local Computer Network, *Communications ACM* 25, 1982, pp. 246-260.

Strasser, M. & Rothermel, K. (2000). System Mechanisms for Partial Rollback of Mobile Agent Execution, *in Proceedings of the 20th International conference on distributed Computing Systems (ICDCS 2000)*, page 20, ISBN: 0-7695-0601-1, IEEE Computer Society, Washington, DC. USA, 2000.

van der Aalst, W.M.P. (1996). Three Good reasons for Using a Petri-net-based Workflow Management System. In *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pp. 179-201, Cambridge, Massachusetts, Nov. 1996.

van der Aalst, W.M.P. (1998). The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, Vol. 8, No. 1, pp. 21-66.

van der Aalst, W.M.P. (1999). Interorganizational Workflows: An Approach based on Message Sequence Charts and Petri Nets. *Systems Analysis-Modelling-Simulation*, Vol. 34, No.3, pp. 335-367.

van der Aalst, W.M.P. & Anyanwu, K. (1999) Inheritance of Interorganizational Workflows to Enable Business-to-Business E-commerce. In *Proceedings of the Second International Conference on Telecommunications and Electronic Commerce (ICTEC'99)*, pp. 141-157, October 1999, Nashville, Tennessee.

van der Aalst, W.M.P. & Hee, K. (2002). *Workflow Management: Models, Methods and Systems*. MIT Press. ISBN: 0-262-72046-9, London.

van der Aalst, W.M.P. & Hofstede, A. (2005). YAWL: Yet Another Workflow Language. *Information Systems*. Vol. 30 No. 4, pp.245-275. ISSN: 0306-4379.

Wang, M.; Wang, H. & Xu, D. (2005), The design of intelligent workflow monitoring with agent technology, *Knowledge-Based Systems*, Vol. 18, Issue 6, pp. 257-266, ISBN: 0950-7051.

Wang, J. (1998), *Timed Petri Nets, Theory and Application*, Kluwer Academic Publishers, (Norwell, Massachusetts), ISBN: 0792382706, pp 281.

WfMC. (1999). Workflow Management Coalition Terminology & Glossary. Technical report, *The Workflow Management Coalition, Document Number WFMC-TC-1011*, available at: http://www.wfmc.org/

Wooldridge, Michael (2001). Intelligent Agents: The Key Concepts, In *Proceedings of the Multi-Agent-Systems and Applications II, (LNCS)*, Vol. 2322, pp 3-43, ISBN:3-540-43377-5, Berling, Heidelberg, 2002, Springer-Verlag, London, UK.

Wooldridge Michael (2002), *An Introduction to multiagent Systems*, John Wiley & Sons (Chichester, England), ISBN: 978-0-471-49691-5. 366 pp.

Xu, J. & Pears, S. (2006). A Dynamic Shadow Approach to Fault-Tolerant Mobile Agents in an Autonomic Environment, *Real-Time Systems*, Vol. 32 No. 3, pp. 235-252, March 2006.

Yan, Y.; Maamar, Z. & Shen, W. (2001). Integration of Workflow and Agent Technology for Business process Management, In *Proceedings of the Sixth international Conference en CSCW in Design*, pp. 420-426, ISBN: 0-660-18493-1, London, Ontario, Canada. July 2001.

Zisman, M.D. (1977). Representation, Specification and Automation of Office Procedures. *PhD Thesis*. Wharton School of Business, University of Pennsylvania. 1977.

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL I.P.N.
UNIDAD GUADALAJARA

"2010, Año de la Patria, Bicentenario del Inicio de la Independencia y Centenario del Inicio de la Revolución"

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó la tesis

Protocolos de tolerancia a fallas en sistemas de agentes móviles para la gestión de flujo de trabajo interorganizacional

del (la) C.

Marina FLORES BADILLO

el día 23 de Agosto de 2010.

Dr. Luis Ernesto López Mellado
Investigador CINVESTAV 3B
CINVESTAV Unidad Guadalajara

Dr. Félix Francisco Ramos Corchado
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Antonio Ramírez Treviño
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Mario Angel Siller González Pico
Investigador CINVESTAV 2A
CINVESTAV Unidad Guadalajara

Dra. María Elena Meda Campaña
Profesor Titular
Universidad de Guadalajara CUCEA

Dr. Víctor Hugo Zaldívar Carrillo
Profesor - Investigador
ITESO