xx(178703.1)

TK1665 .G8
.T68
2009

Centro de Investigación y de Estudios Avanzados

del I.P.N.

Unidad Guadalajara

Simulación Distribuida de Tráfico Urbano basada en Sistemas Multi-Agente

Tesis que presenta:
Emmanuel López Neri

para obtener el grado de:
Doctor en Ciencias

en la especialidad de:
Ingeniería Eléctrica

Asesores de Tesis:
Dr. Luis Ernesto López Mellado
Dr. Antonio Ramírez Treviño

Guadalajara, Jalisco, Octubre 2009.

Simulación Distribuida de Tráfico Urbano basada en Sistemas Multi-Agente

Tesis de Doctor en Ciencias
En Ingeniería Eléctrica

Por:
Emmanuel López Neri
Ingeniero Electrónico en Computación
Centro de Enseñanza Técnica Industrial 1998-2003

Beca otorgada por CONACYT, No. 180892

Asesores de Tesis:
Dr. Luis Ernesto López Mellado
Dr. Antonio Ramírez Treviño

CINVESTAV del IPN Unidad Guadalajara, Octubre , 2009.

Contenido

Agradecimientos:

A mis padres y hermanas por su apoyo incondicional,
A Chío por su soporte constante y cariño,
Mis más sinceros agradecimientos a mis colegas y amigos,
A mis asesores por su asesoramiento científico y estímulo
para seguir creciendo intelectualmente,
Al ***CONACYT*** por su apoyo.

Becario de CONACYT No. 180892

Resumen

En esta tesis se aborda el problema de la simulación distribuida de sistemas de tráfico urbano (STU); el enfoque adoptado es el de la micro-simulación orientada a eventos, basada en sistemas de agentes móviles. Este trabajo contiene dos contribuciones principales: una metodología de modelado de STU y una arquitectura para la simulación distribuida, las cuales están soportadas por una técnica de implementación basada en la arquitectura propuesta.

La metodología de modelado permite capturar la información necesaria para simular el STU. Una parte importante de la descripción del UTS es soportada por un modelo de tres niveles en n-LNS el cual describe principalmente la red vial, el comportamiento general de los vehículos (representados por agentes móviles) y las actividades a realizar por éstos. La parte complementaria del modelo está dada por las bases de conocimiento que soportan los procesos de razonamiento de los agentes para la toma de decisiones.

La arquitectura propuesta permite la ejecución del modelo en nLNS asociando el disparo de sus transiciones con una estructura de eventos futuros (E^2F) que captura las restricciones de causalidad existentes en los STU. En base a esta estructura es propuesta una estrategia para ejecutar eventos con la misma marca de tiempo en el mismo paso de simulación; esto permite relajar las restricciones de causalidad del enfoque de procesos lógicos (PL), comúnmente usados para la simulación paralela y distribuida de sistemas de eventos discretos.

En el desarrollo del simulador se ha requerido la construcción de una biblioteca, desarrollada en Java, para implementar los componentes del modelo multi-agente del STU. La ejecución del simulador, soportada por el middleware Pro-Active ha permitido probar varios casos de estudio donde se analiza el desempeño de diversas políticas de control de tráfico.

Urban Traffic Distributed Simulation Based on Multi Agent Systems

Abstract

In this thesis the problem of distributed simulation of Urban Traffic Systems (UTS) is addressed; the approach held is mobile agent based event oriented micro-simulation. This work includes two main contributions: a modeling methodology for UTS and architecture for distributed simulation, which are supported by an implementation technique regarding the proposed architecture.

The modeling methodology allows capturing the information needed to simulate the UTS. An important part of the UTS description is supported by a three level model expressed in nLNS which describes mainly the traffic network, the general behavior of vehicles (represented by mobile agents), and the activities to perform by the vehicles. The complementary part of the model is given by the knowledge bases supporting the agents reasoning processes for decision making.

The proposed architecture allows the execution of the nLNS model by associating the transitions firing to a future event structure (E^2F) which captures the causality relations of the UTS. Based on this structure, a strategy for executing events with the same timestamp in the same simulation step is proposed; this allows avoiding the causality constraints from the Logical Processes (LP) approach, widely used in parallel and distributed discrete event simulation.

The developed simulator includes a Java based library specially built for implementing the components of the multi-agent model of the UTS. The simulator execution, supported by the middleware Pro-Active, allowed to test several case studies in which the performance of diverse traffic control policies has been analyzed.

Introducción

El crecimiento en los últimos años del tráfico urbano es ya un problema global. Los problemas de congestión y capacidad vial son causantes de demoras que perjudican a los usuarios y a la economía de los países que la padecen [1]. El tráfico urbano también es fuente de problemas ambientales originados por el ruido y la contaminación [2][3]. La solución a estos problemas se enfoca comúnmente en el diseño y construcción de calles y avenidas. Sin embargo en la actualidad se investiga en estrategias de control eficientes, tales como la aplicación de los sistemas de transporte inteligente (STI) que permiten un uso racional de la infraestructura vial actual.

Los STI surgen como consecuencia de la aplicación de las tecnologías de telecomunicaciones, informática, electrónica, sensores y técnicas de procesamiento, almacenamiento y visualización de la información, en los sectores de vialidad y transporte. Entre los dispositivos que permiten monitorear y controlar el STU se encuentran: señalamientos de mensaje variable, radio informante en carreteras, señalamiento de límite de velocidad variable, sensores de tráfico, etc. Sin embargo, debido al elevado costo de estos dispositivos, su uso es limitado.

Para predecir la eficiencia de estrategias de control de tráfico aplicadas en un STU real, los ingenieros de transporte y analistas de planeación urbana han hecho uso de la simulación como herramienta de soporte en la evaluación de desempeño de estos dispositivos, ya que permite evaluar el desempeño de estrategias de control mediante el análisis del flujo vehicular. También, la simulación puede ser parte de las tareas de predicción de flujo vehicular de un STI, o bien como soporte en el diseño de nueva infraestructura vial.

El nivel de detalle requerido por los simuladores actuales es microscópico, ya que se requiere distinguir cada entidad del STU con sus propiedades y comportamientos individuales. Actualmente este nivel de detalle ha sido capturado por diferentes enfoques de modelado: modelos de vehículo-siguiente (*car-following*) [4][5][6][7], los autómatas celulares [8][9] [10] y teoría de colas [11]. La mayoría de estos modelos usa la técnica de simulación de tiempo discreto (TD) para realizar el avance de la simulación en un lapso de tiempo predefinido. Sin embargo en estos modelos no se permite capturar la capacidad de reacción y percepción de los conductores en el STU real.

Los SMA son un paradigma de programación que proporciona los medios para modelar a cada conductor como una entidad con características únicas, capacidad de razonamiento y percepción del ambiente en el que se encuentra inmerso. Actualmente existen varios trabajos representativos del uso de este enfoque para simular de manera eficaz STU reales utilizando la técnica TD para realizar el avance de la simulación [12] [13] [14] [15] [16][17], a pesar de los elevados recursos computacionales que requiere.

Algunos modelos utilizan la técnica de simulación de eventos discretos (ED). En esta técnica la ejecución de un evento describe un cambio en el estado del sistema y por lo tanto no existe un intervalo de incremento del tiempo de simulación fijo, como el caso de TD. Aunque ha sido probado que la técnica de ED es más eficiente que la de TD [18] [19][20], ésta solo ha sido usada en los modelos de teoría de colas para describir la salida de los vehículos en las intersecciones [21], en los modelos bajo el enfoque SMA para describir la coordinación de estrategias de control de tráfico urbano [22][23][24] y para distribuir la simulación bajo el enfoque de simulación paralela y distribuida de eventos discretos (SPED). El enfoque SPED es ampliamente usado para obtener una simulación descentralizada y distribuida de modelos ED. En este enfoque, el modelo se divide en un conjunto de procesos lógicos (PL), cada uno manteniendo y

procesando una porción del ambiente del sistema. Los cambios de estado son modelados como eventos con marca de tiempo (*time-stamp)*.

De acuerdo a la revisión de literatura realizada, los enfoques existentes para el desarrollo de simuladores parecen ser *ad hoc* y no existe una metodología sistemática para su construcción. Se puede notar que la principal causa de la ausencia de modelos de STU bajo el enfoque de SMA usando la técnica de avance de simulación de ED, es la complejidad al capturar múltiples relaciones entre las diferentes entidades y los procesos del sistema.

En esta tesis se aborda el problema de la simulación distribuida de sistemas de tráfico urbano (STU); el enfoque adoptado es el de la micro-simulación orientada a eventos, basada en sistemas de agentes móviles. Este trabajo contiene dos contribuciones principales: una metodología de modelado de STU y una arquitectura para la simulación distribuida, las cuales están soportadas por una técnica de implementación basada en la arquitectura propuesta.

La metodología de modelado permite capturar la información necesaria para simular el STU. Una parte importante de la descripción del UTS es soportada por un modelo de tres niveles en n-LNS el cual describe principalmente la red vial, el comportamiento general de los vehículos (representados por agentes móviles) y las actividades a realizar por éstos. La parte complementaria del modelo está dada por las bases de conocimiento que soportan los procesos de razonamiento de los agentes para la toma de decisiones.

La arquitectura propuesta permite la ejecución del modelo en nLNS asociando el disparo de sus transiciones con una estructura de eventos futuros (E^2F) que captura las restricciones de causalidad existentes en los STU. En base a esta estructura es propuesta una estrategia para ejecutar eventos con la misma marca de tiempo en el mismo paso de simulación; esto permite relajar las restricciones de causalidad del enfoque de procesos lógicos (PL), comúnmente usados para la simulación paralela y distribuida de sistemas de eventos discretos.

En el desarrollo del simulador se ha requerido la construcción de una biblioteca, desarrollada en Java, para implementar los componentes del modelo multi-agente del STU. La ejecución del simulador, soportada por el middleware Pro-Active ha permitido probar varios casos de estudio donde se analiza el desempeño de diversas políticas de control de tráfico.

Esta tesis está organizada como sigue: En el capítulo 1 se presentan los conceptos de simulación de eventos discretos y simulación basada en sistemas multi-agentes, así como una revisión de literatura de la simulación de sistemas de tráfico urbano. En el capítulo 2 se describe la metodología de modelado propuesta para STU utilizando el formalismo n-LNS y su correspondencia con los SMA. En el capítulo 3 se describe la arquitectura de simulación distribuida para el modelo de STU y los mecanismos de ejecución concurrente y distribuida propuestos. En el capítulo 4 se presentan los aspectos importantes de la implementación y resultados para la validación del modelo. Finalmente, en las conclusiones se resaltan las características de la metodología y arquitectura propuestas y se comentan las extensiones inmediatas y a largo plazo del presente trabajo.

CAPITULO 1

MODELADO Y SIMULACIÓN DE SISTEMAS DE TRÁFICO URBANO

En este capítulo algunas de las clasificaciones básicas de métodos de modelado de los sistemas de tráfico urbano serán introducidas, mientras que las propiedades más importantes de cada modelo son explicadas. Cabe recalcar que la clasificación presentada resalta la capacidad de cada modelo al usar alguna de las técnicas para realizar el avance de la simulación (dirigida por eventos o por tiempo). Se mostrarán algunos ejemplos de cada modelo y finalmente se presentará una discusión y la propuesta de este trabajo de investigación.

1.1. Sistemas de tráfico urbano

Aunque los sistemas de tráfico urbano (STU) son bien conocidos por todos, ya que en la vida cotidiana se tiene relación directa o indirecta con éstos, y aunque la mayoría de los trabajos pasen por alto una definición explícita, creemos conveniente definirlo para establecer los límites y el contexto utilizado en este trabajo de investigación. A continuación se presenta una definición de STU:

Definición 1.1: Un STU (ver Fig. 1.1) es un sistema complejo no-lineal y variante en el tiempo compuesto de una estructura de red (calles, avenidas, carreteras, etc.), entidades con comportamiento propio (peatones, conductores, viajantes, etc.), objetos (arboles, señalamientos, obstáculos, etc.), políticas de tránsito y leyes físicas propias del sistema. Además, de dos sub-sistemas: generación demográfica y control de tráfico.

La estructura de red vial es un conjunto de vías interconectadas formando intersecciones; las cuales son utilizadas por las entidades para trasladarse de un punto a otro de la red, usando distintos medios de transporte (autobús, motocicleta, taxi, metro, vehículo, etc.) y para contener los objetos que pueden ser obstáculos o funcionar como objetos informativos para las entidades (señalamientos de velocidad, de prioridad de cruce, alto total, cruce de peatones, etc.) con el objetivo de regular su comportamiento. La interacción de una entidad con otra, o con los objetos, origina comportamientos complejos conocidos como emergentes (comportamientos colectivos o no triviales). En ocasiones son comportamientos inesperados, que sólo se pueden observar en cantidades macroscópicas, tales como: colas, embotellamiento de tráfico, entre otros, de los que se pueden obtener algunos índices de desempeño del sistema (combustible usado, CO_2 emitido, longitud de colas, etc.). Este tipo de sistemas no cuenta con un objetivo global (como en los sistemas de manufactura), si no que cada entidad tiene su propio objetivo (plan de viaje).

Debido al elevado volumen de entidades circulando por la infraestructura vial, se hace imprescindible la existencia de ciertas políticas de tránsito o de comportamiento (PC), que permiten la convivencia entre las entidades. Estas políticas pueden llegar a ser ignoradas por las entidades de acuerdo a ciertos parámetros de comportamiento, tal como el nivel de agresividad o negligencia de la entidad. A continuación se listan algunas de estas políticas:

- No manejar a velocidades mayores a las permitidas en el segmento.
- Manejar por el lado derecho de la calle siempre que sea posible.
- Manejar en la dirección permitida por el segmento.
- Los vehículos deben utilizar solo aquellos segmentos asignados para su tipo en la red de tráfico (autobús, bicicleta, etc.).

La entidad también deberá de observar las restricciones físicas impuestas por el sistema, tales como no poder avanzar si existe otra entidad delante de ésta, o no poder volar o brincar, etc. Este tipo de restricciones, son llamadas leyes de evolución (LE), y no pueden ser ignoradas por las entidades:

- *Ley exclusión mutua.* Dos entidades no pueden ocupar el mismo lugar al mismo tiempo.
- *Ley de entidad ignorada.* Una entidad no puede pasar a través de otra entidad.

- *Ley de negligencia.* Si alguna de las entidades intenta romper alguna de las reglas anteriores, entonces el resultado es un choque entre las entidades involucradas.

Aunado a esto, se requieren sistemas que procuren mejorar la circulación en todo el sistema. Estos sistemas son conocidos como centros de control de tráfico, en los cuales un ordenador (o inclusive un operador) establece en cada momento cuál es la política de control adecuada de acuerdo a las circunstancias del tráfico, implementándola a través de los objetos contenidos en la estructura vial (semáforos, barreras automatizadas, señalamientos de mensaje variable, etc.); actualmente se ha hecho uso de los STI, los cuales buscan optimizar los STU, utilizando dispositivos electrónicos para obtener información en tiempo real y mostrar a las entidades rutas alternas o sugerencias de viaje a través de dispositivos especiales (paneles visuales, control de mensajes variables, cámaras, sensores, peaje automático, etc.).

Además, cada STU tiene características propias de tipo poblacional o demográfica tales como: tipo de viviendas, ubicación de centros de trabajo, número de habitantes, etc. Esta información es generada a través de modelos demográficos **[25]** **[26]**, los cuales utilizan características socioeconómicas de la población, para obtener relaciones que brinden información para su construcción.

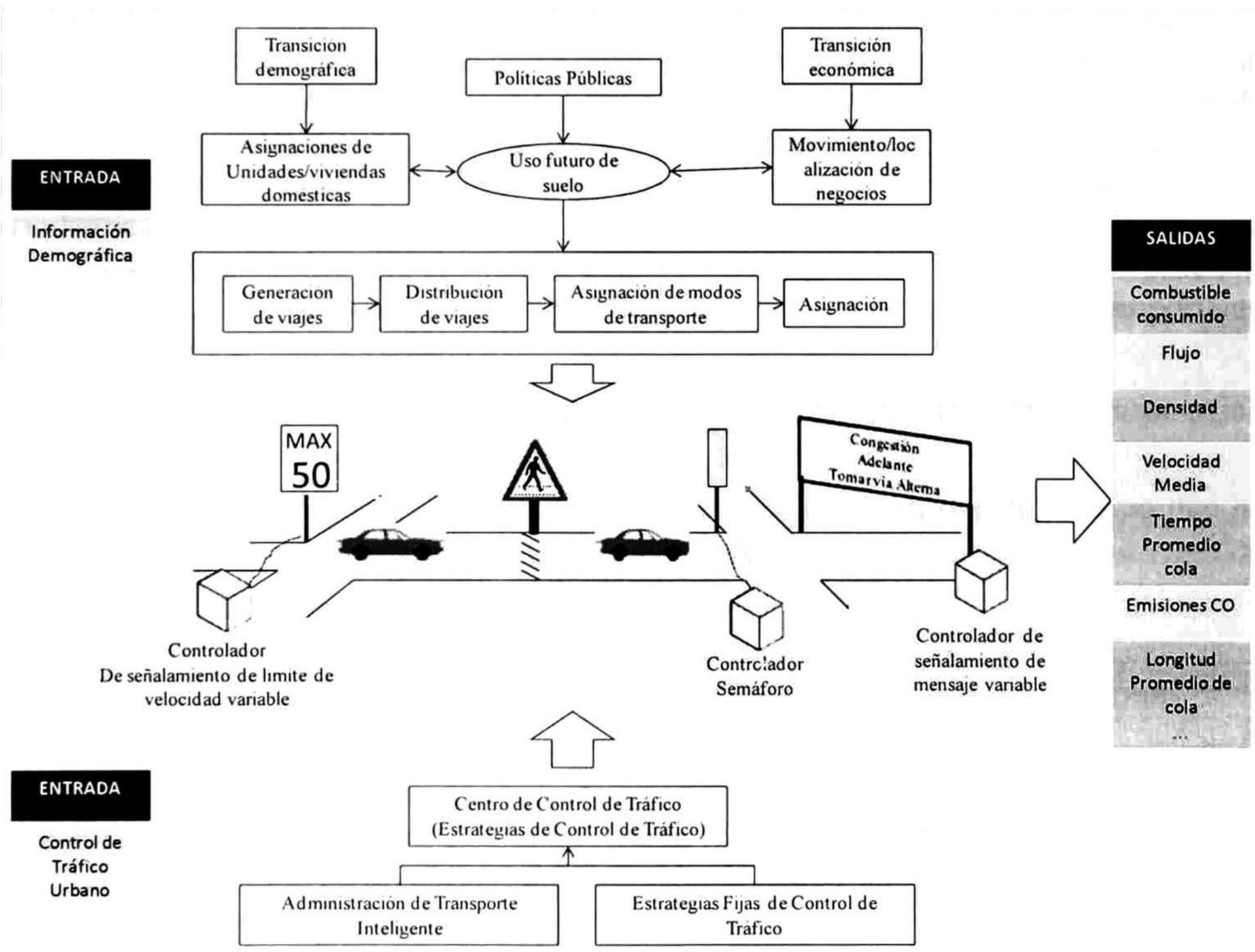


Fig. 1.1: Un Sistema de Tráfico Urbano

1.2. Modelado y simulación de sistemas de tráfico urbano

Durante los últimos 50 años el modelado y simulación de STU ha sido un área de investigación en constante desarrollo. Desde el modelo inicial de Lighthill-Whitham **[27]**, hasta los modelos más recientes físico-psicológicos **[18][28]** y de autómatas

celulares (AC) [**10**], se han propuesto diferentes modelos y una gran cantidad de teorías del flujo vehicular. Sin embargo, es conveniente resaltar que la aparición de todos estos modelos y las distintas clasificaciones existentes, impide tener una visión clara de las ventajas y desventajas de cada modelo. Definir el concepto de "modelo" ayudará a comprender la utilidad de muchos de estos instrumentos de análisis y comprender la clasificación propuesta en este trabajo.

Colomer [**29**], define un modelo como: "representación abstracta, simplificada e inteligible de un aspecto de una realidad, obtenida a través de la observación, validada posteriormente, y que permite comprender mejor la realidad.". La simplicidad del modelo permite comprender con mayor facilidad la realidad que intentar hacerlo a través del análisis directo. Por su parte, Izquierdo [**30**], indica: "normalmente y a excepción de algunos modelos específicos, como son los modelos conceptuales en los que la descripción de la realidad se realiza en términos lógicos sin utilizar formulaciones matemáticas, los modelos tienen una estructura matemática que relaciona los elementos y factores que caracterizan el fenómeno que se analiza. Se trata, por consiguiente, de la representación de un sistema."

El objetivo de la mayoría de los modelos que se han propuesto es el control del tráfico en las grandes ciudades. El tamaño del problema y el tipo de comportamiento, claramente no lineal de todos los elementos implicados, hace que sea muy complejo conocer *a priori* qué implicaciones puede tener el utilizar una u otra estrategia de control. Además, antes de poner en práctica cualquier iniciativa de control, es necesario poder estudiar en un entorno virtual el posible impacto que pueda tener en el sistema. Estos estudios se apoyan normalmente en técnicas de simulación y optimización de tráfico, herramientas que permiten al ingeniero de tráfico determinar los parámetros de control adecuados implementados en ciertos dispositivos tales como los semáforos o señalamientos de mensaje variable.

La simulación es una alternativa válida para el análisis y evaluación de sistemas [**31**], cuando es complejo aplicar herramientas analíticas. En la simulación es importante comprender los siguientes conceptos:

- El *Tiempo de Simulación*, que representa el valor del tiempo transcurrido en la operación del sistema real.
- El *Tiempo de Ejecución* (*wall-clock*), que es el valor del tiempo que ha transcurrido durante la ejecución de la simulación. A veces también llamado.
- Un *Paso Simulación* puede ser definido como el momento discreto en el que una acción toma lugar y la información necesaria es actualizada.

Con el fin de explicar el comportamiento del flujo del tránsito vehicular se han tomado en cuenta áreas del conocimiento como matemáticas, física y computación; en las cuales se establecen diversas teorías: Teoría Cinética de Gases, Teoría de Colas, Teoría de Dinámica de Fluidos, etc. El objetivo es utilizar los principios generales de las teorías antes mencionadas con el fin de establecer modelos los cuales permitan describir el comportamiento del tránsito tomando en consideración diferentes escenarios. La teoría de flujo vehicular busca describir de manera matemática las interacciones entre vehículos que viajan a través de la red de tráfico. Para hacerlo, es necesario tomar en consideración una serie de variables y parámetros que determinan la reacción del conductor y de su vehículo ante una determinada situación. Las principales variables a considerar son la concentración o densidad, que tiene que ver con el número de

vehículos en un espacio determinado de la estructura vial en un tiempo dado, el flujo, que tiene que ver con la forma en que se desplazan los vehículos a través de la estructura vial en un cierto número de pasos de tiempo, y por último, la velocidad, que tiene que ver con la rapidez con la que se desplazan los vehículos en las calles. Estos modelos conjugan elementos estocásticos y determinísticos y pueden ser de tipo macroscópico o microscópico, según el nivel de detalle de la simulación.

Los modelos de simulación macroscópica describen el tráfico a un nivel de agregación alto, como un flujo, sin distinguir las partes que le constituyen. Por ejemplo, el flujo de tráfico es representado de una manera agregada, usando características tales como nivel de flujo, densidad y velocidad. Las maniobras de los vehículos individuales, tales como los cambios de carril, no son por lo general representadas de manera explícita. En un modelo de simulación macroscópico se puede suponer que el flujo de tráfico está asignado de manera propia a los carriles de las avenidas y se emplea un método de aproximación para éste fin. Los modelos de flujo macroscópico pueden ser clasificados de acuerdo al número y orden de ecuaciones parciales que forman parte del modelo.

En la simulación microscópica se considera el movimiento de cada vehículo de forma individual, calculan a cada paso de simulación su posición, velocidad y aceleración. Estos modelos requieren una gran cantidad de datos, los modelos que utiliza son bastante complejos y los costos computacionales requeridos son muy elevados. Por el contrario ofrece resultados que dan una idea muy detallada del funcionamiento del STU y es muy útil para analizar estrategias de control. Debido al aumento de la capacidad computacional y la mejora de técnicas para resolver problemas de control a gran escala, se ha optado por el uso de los modelos microscópicos. En este trabajo estamos interesados en este tipo de simulación.

Casi todos los modelos de simulación de tráfico describen sistemas dinámicos donde el tiempo es siempre la variable independiente básica. Los modelos de simulación continuos describen el cambio de estado de los elementos de un sistema de manera continua a lo largo del tiempo en respuesta a un estímulo continuo; es el caso de los modelos propuestos por Lighthill-Whitham [**27**]. Los modelos de simulación discreta representan los sistemas asegurando que sus estados cambien abruptamente en puntos específicos en el tiempo. Existen dos tipos de modelos discretos: TD y ED.

El primero, segmenta el tiempo en una sucesión de intervalos previamente establecidos. En cada intervalo, el modelo de simulación calcula las actividades que cambiarán los estados de los elementos seleccionados del sistema. Esta técnica es análoga a representar un valor inicial de una ecuación diferencial en la forma de una expresión de diferencia finita con la variable independiente Δt.

Algunos sistemas se caracterizan por tener entidades "ociosas" por mucho tiempo. Por ejemplo, el estado de un semáforo en la intersección de un STU. Digamos que se encuentra en verde, permanecerá en ese estado por varios segundos hasta que su estado cambie instantáneamente a amarillo. Este cambio abrupto en el estado es llamado evento. Ya que es posible describir con precisión la operación del semáforo registrando sus cambios de estado como una sucesión de eventos temporizados, se puede tener un ahorro considerable en los tiempos de ejecución de la simulación o de recursos computacionales al solo ejecutar aquellos eventos que den lugar a un cambio de estado en lugar de calcular el estado del semáforo segundo a segundo.

A continuación, es presentada una revisión del modelado y simulación de STU; se presenta para cada enfoque un ejemplo de implementación utilizando la técnica de avance de simulación TD o ED. Inmediatamente después se hará un análisis de ventajas y desventajas de cada modelo, se discutirá sobre el enfoque seleccionado para este

trabajo de investigación, así como los retos existentes en dicho enfoque, los cuales son las guías de trabajo de esta tesis.

1.2.1. Modelos de seguimiento de vehículos

Actualmente existen varias propuestas de modelos de seguimiento de vehículos [4][5][6][7][32][33][8] [34], que inclusive incorporan comportamientos tales como espacio-disponible y cambio-carril [35][36][37][38]. Estos modelos son clasificados en dos grupos, los modelos de "Respuesta a Estímulos" y los modelos de "Distancia Segura" o "Distancia de Seguridad". El primer grupo considera la aceleración del vehículo siguiente como una función entre la diferencia de velocidades (entre el vehículo líder y el vehículo siguiente) y el tiempo de reacción del conductor.

Los modelos de distancia segura consideran la aceleración del vehículo siguiente como una relación entre las diferencias de velocidades (del líder y seguidor) y la distancia que hay entre ambos procurando siempre mantener una distancia mínima que evite la colisión entre ambos vehículos en caso de un frenado brusco. Posiblemente el más utilizado es el propuesto por Gipps [39] que pertenece a éste último grupo.

AIMSUN(modelo TD)

AISUM [40] es un software comercial para la simulación de tráfico, y será usado como modelo representativo de aquellos que utilizan los modelos de seguimiento de vehículos para realizar sus simulaciones. Con este enfoque, generalmente el paso de simulación es de 1 segundo, tiempo en el que las salidas del modelo (aceleración, desaceleración, cambio de carril, etc.) de cada vehículo son recalculadas y los modelos de las tareas de conducción individuales (vehículo-seguidor, cambio-carril, espacio-disponible, etc.) son actualizados. AIMSUN utiliza una versión del modelo de Gipps de seguimiento de vehículos. Con el objetivo de dar claridad en la exposición del modelo se usará como apoyo la Fig. 1.2.

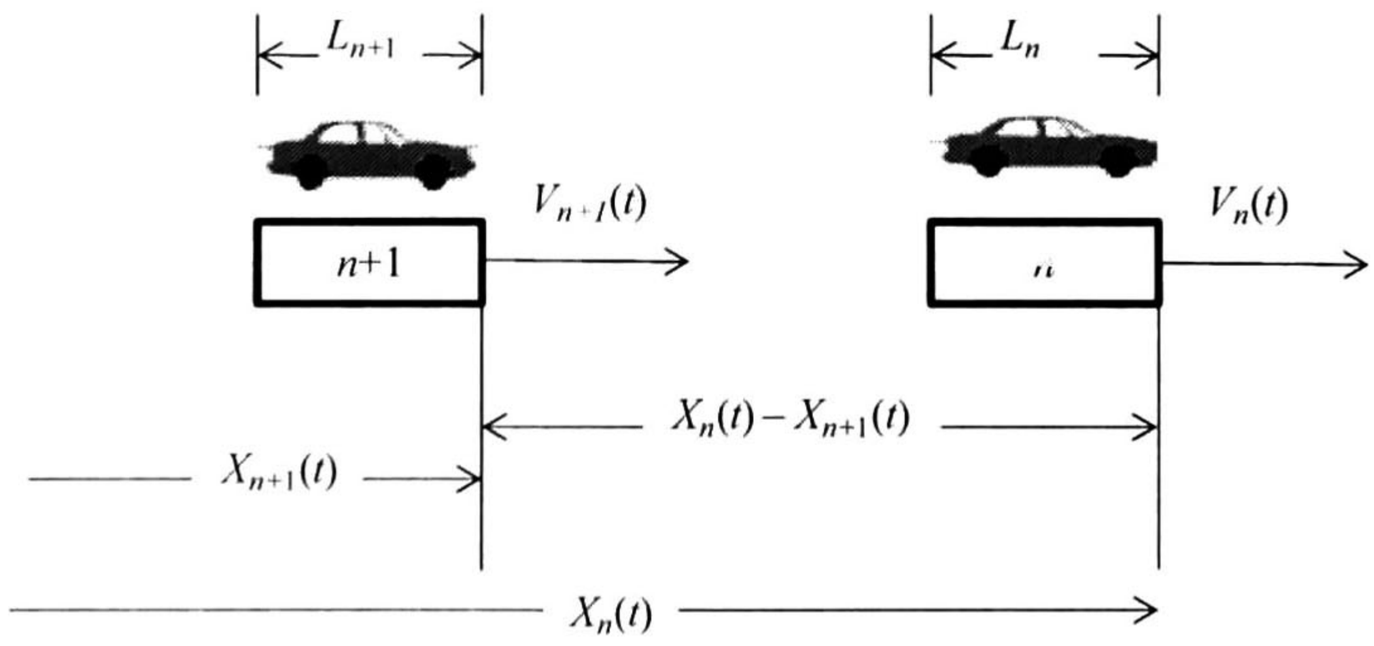


Fig. 1.2: Modelo de Gipps de vehículo Siguiente

Considerando que la dirección del flujo es de izquierda a derecha, n representa al vehículo líder con una longitud de L_n y una velocidad en el instante t de $V_n(t)$. Mientras que n +1 representa al vehículo seguidor con una longitud de L_{n+1} y con una velocidad de $V_{n+1}(t)$ para el mismo instante de tiempo t. La función $X_n(t)$ es la posición del vehículo líder en el instante t y así también $X_{n+1}(t)$ corresponde a la posición del vehículo siguiente para una misma t La diferencia $X_n(t)$-$X_{n+1}(t)$ es la distancia existente entre los dos vehículos sin considerar la longitud del vehículo líder.

El modelo de Gipps considera la presencia de un vehículo líder directamente delante del vehículo seguidor y condiciona el comportamiento de éste último. Factores como la velocidad del líder, su desaceleración, la distancia entre ambos vehículos y el tiempo de reacción condicional, determinan el comportamiento del vehículo siguiente y la velocidad que éste desarrollará. El modelo matemático es el siguiente:

$$V_{n+1}(t+T) = d_{n+1} + \sqrt{d_{n+1}^2 * T^2 - d_{n+1}\left[2 * \{X_n(t) - S_n - X_{n+1}(t)\} - V_{n+1} * T - \left(\frac{V_n^2(t)}{d_n}\right)\right]} \quad \textbf{(Ecn. 1.1)}$$

Donde:

- d_{n+1} es la máxima desaceleración que dese aplicar el conductor del vehículo siguiente. $d_{n+1} < 0$.
- $X_n(t)$ es la posición del vehículo *n* en el instante *t*.
- S_n corresponde al tamaño efectivo del vehículo *n*, esto es, el largo del vehículo líder L_n (ver Fig. 1.2) más una distancia o espacio que el vehículo siguiente no podrá invadir (una distancia de seguridad).
- $X_{n+1}(t)$ es la posición del vehículo siguiente en el momento t.
- d_n es la desaceleración deseada por el vehículo líder. $d_n < 0$.

La diferencia bajo la raíz cuadrada y entre llaves $\{X_n(t) - S_n - X_{n+1}(t)\}$ corresponde a la distancia que existe en el instante *t* entre los dos vehículos considerando el tamaño efectivo del líder. Esta distancia determinará si el vehículo siguiente obtiene su velocidad desde la variante de flujo libre (no hay vehículo líder o está muy retirado) o de la condicionada. Este modelo es ejecutado para todas las entidades del sistema en cada paso de simulación.

SCATTER (modelo ED)

Perumalla y sus colegas [**41**][**42**] avanzan la simulación usando la técnica de ED. La idea principal es que los eventos importantes en el movimiento vehicular se pueden capturar fácilmente por un proceso en el nodo intersección del STU. Es decir, en cada intersección, se ejecuta un algoritmo que se encarga de modificar el estado de los vehículos contenidos en los segmentos conectados a dicha intersección. Para cada vehículo se ejecuta el modelo de vehículo-siguiente y el de cambio-carril. Existe un mecanismo de ejecución de eventos discretos que ejecuta aquellos eventos cuya marca de tiempo se igual a la del tiempo de simulación. Los vehículos, contenidos en cada carril conectado a la intersección, son avanzados por un paso de simulación igual al tiempo actual menos el último instante en el cual fueron actualizados. Cuando los vehículos estén listos para salir, ya sea de la cola de vehículos en la intersección o de los carriles conectados a la intersección, se aplica un proceso para incorporar los efectos de congestión y control de tráfico, de manera que pueda tomarse una decisión con respecto al tiempo del evento de salida del vehículo de la intersección.

1.2.2. Autómatas celulares

Los autómatas celulares (AC) son una técnica de modelado de cómputo que consiste en 'discretizar' el espacio en celdas de igual tamaño; cada uno de éstas

ocupadas en cada paso de simulación solo por un vehículo a la vez. Los vehículos se desplazan a través de estas celdas cambiando de su velocidad de acuerdo al espacio disponible delante de ellos. La evolución temporal del estado de las celdas es ejecutada de manera síncrona de acuerdo a reglas de vecindad local. Los AC son simples y mantienen una resolución espacial considerable.

Existen varios modelos de STU utilizando este técnica [**8**][**9**][**10**][**43**]. En los modelos microscópicos de simulación de tráfico basados en AC, el tamaño que representa cada celda del autómata y del paso de simulación, i.e., la resolución espacial y temporal de la simulación son dos parámetros clave durante la ejecución de las simulaciones. Para la mayoría de los modelos de simulación de tráfico microscópico basados en AC, la selección del tamaño de la celda del AC y del paso de simulación son un poco arbitrarios. En TRANSIMS[**44**], el tamaño de la celda del AC es de 7.5m, mientras que el paso de simulación es de 1s. En [**45**], el tamaño del AC es establecido a 5m, y el tiempo del paso de simulación también es igual a 1s. Aparentemente, ambas resoluciones (espacial y temporal) son bajas

Por ejemplo, en TRANSIMS, si un vehículo es conducido a una velocidad de 4 celdas por segundo, la diferencia de velocidad entre 95 km/h (3.5 * 7.5m/s) y 121 km/h (4.4999 * 7.5 m/s) no será distinguido por los modelos de simulación. Todos los vehículos con la velocidad dentro de ese rango se moverán a la misma velocidad sin ninguna diferencia. Con respecto a la resolución temporal, si el paso de simulación es definido uniformemente a 1 segundo, los modelos de simulación difícilmente mostrarán con exactitud diferencias entre los conductores tales como tiempo de reacción, retardo de arranque a la intersección, entre otros. La mayoría de los trabajos actuales usando AC, modelan sistemas de tráfico de alta velocidad (carreteras, autopistas, etc.) en lugar de STU. Por la caracterización de los AC no existe un modelo que se ejecute con la técnica de avance de simulación de ED.

1.2.3. Teoría de colas

Si la velocidad de simulación del modelo no es suficiente para cierto tipo de aplicaciones, una solución es cambiar a un modelo mucho más simple. Con respecto a la velocidad de simulación, los modelos basados en teoría de colas logran un alto desempeño. Aquí, la suposición básica es que las principales características de un STU pueden ser modeladas observando las intersecciones de manera individual. Esta suposición se basa al observar que en las redes de tráfico urbano:

- El tráfico fluye más o menos de manera libre en los segmentos.
- Los vehículos se acumulan en la entrada de la intersección siguiente y el vehículo espera que los vehículos al frente se muevan.

La teoría de colas es una de las herramientas que con mayor frecuencia se usan en el análisis de problemas de congestionamiento vial. En los modelos de tráfico vehicular, los fenómenos de retardo, tales como la reducción de carriles, casetas de cobro, semáforos, etc., pueden ser tratados como estaciones de servicio. Para entender algunas de las ecuaciones más importantes, se definen las siguientes variables:

- λ : Representa la tasa de llegadas de vehículos a una estación de servicio.
- μ : Representa la tasa media de servicio a la que se atienden los vehículos de la cola

La tasa de llegadas es también llamada demanda, así como la tasa de servicio es llamada capacidad. La disciplina que rige una cola que simula cualquier fenómeno de tráfico vehicular, es la más común de la teoría de colàs, y se enuncia como: "El primero que llega es el primero que sale."

La capacidad de un semáforo tiene en el flujo de saturación su valor máximo, y será igual a μ siempre y cuando no se presente una cola usando este valor. Es decir, si los vehículos llegan dispersos ó bajo flujo libre a una intersección semaforizada, $\lambda = \mu$; o en otras palabras, los vehículos son atendidos (salen) a la misma tasa con que llegan.

MATSIM-T (TD)

Varios modelos utilizan la teoría de colas para representar el flujo vehicular [**11**]. Usaremos el trabajo desarrollado por Cetin y sus colaboradores [**46**] llamado MATSIM-T. En este trabajo se asume que las características de un STU pueden ser capturadas en las intersecciones, resultando un modelo donde:

- El recorrido de los vehículos a través de la red de tráfico ya no es simulada de manera directa; en lugar de eso se cambia la perspectiva y se simulan los segmentos (calles) entre intersecciones. Es decir los segmentos son simulados como si estos "procesaran" a los vehículos que navegan en la red.
- Una cola almacena los vehículos que entran en cada segmento, así como su respectivo tiempo de entrada.
- Las capacidades de cada segmento y el espacio disponible para los vehículos son parámetros de entrada del modelo.
- Cuando un vehículo desea dejar el segmento A para entrar al segmento B, en el siguiente paso de simulación, el segmento A verifica que el vehículo pueda dejar tal segmento verificando varias restricciones (capacidad, tiempo de viaje con velocidad libre, intersección precedente, y el espacio disponible en el siguiente segmento). Si todas las precondiciones se cumplen, entonces el vehículo puede dejar el segmento A y entrar al segmento B.

MATSIM-T (ED)

En el modelo del simulador MATSIM-T [**46**], cada segmento es procesado en cada paso de simulación, esto genera ineficiencias en el desempeño del simulador para segmentos con baja densidad; ya que la mayor parte del tiempo ningún vehículo es procesado debido a que su tiempo de salida de la cola no se ha cumplido.

Nagel y sus colegas [**21**] sustituyen el tiempo constante por eventos que describen las acciones de los vehículos en cada intersección. De esta manera solo cuando el vehículo está listo para salir entonces su evento es procesado. En la Fig. 1.3 se muestra el protocolo de interacción de los vehículos en la intersección.

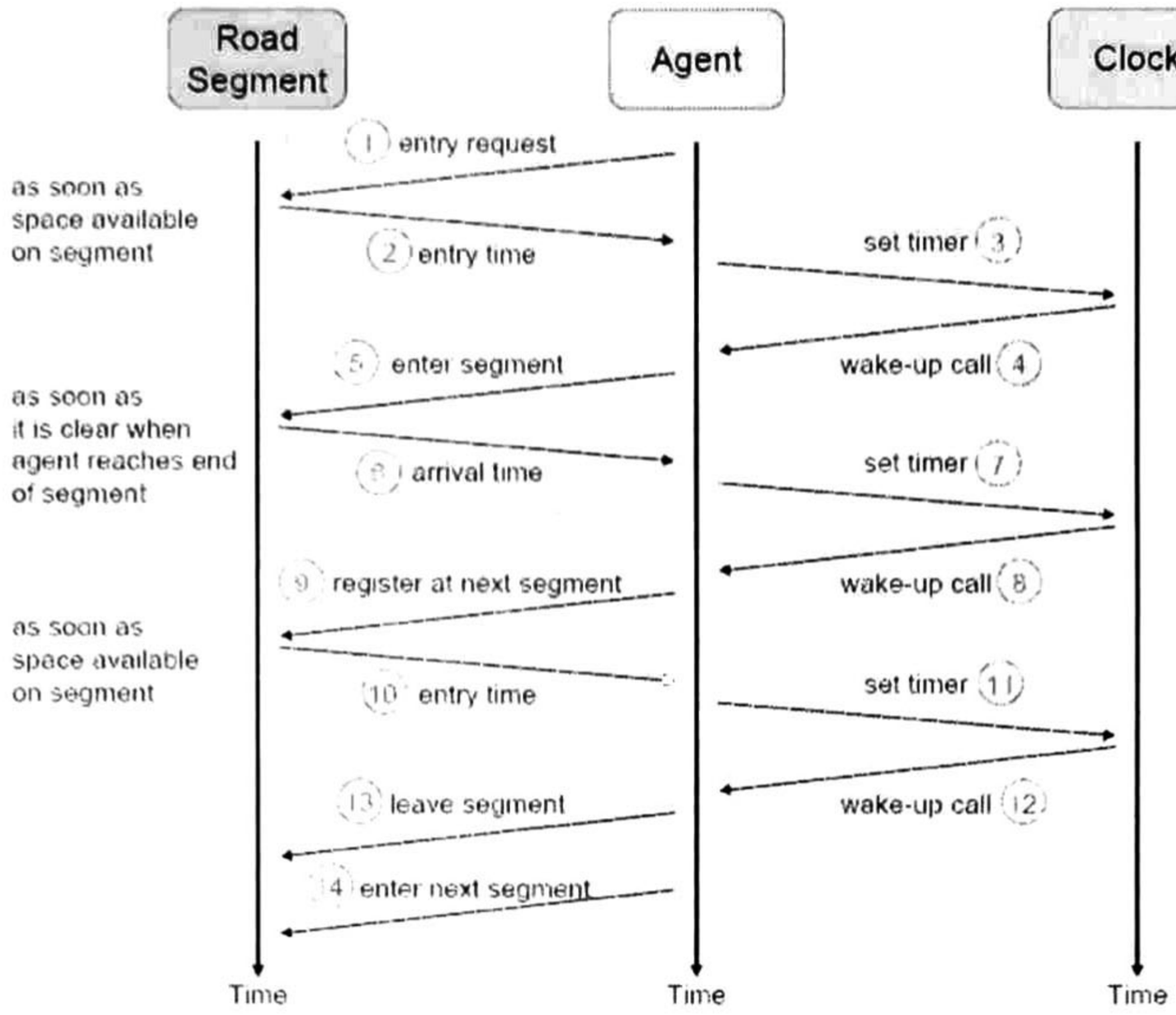


Fig. 1.3: Protocolo de Flujo de Tráfico

1.2.4. Redes de Petri

Un formalismo que permite capturar modelos de ED son las redes de Petri (RP). En la literatura existen varios modelos de STU descritos con RP **[47][48][49][50]**. Su principal aplicación ha sido en el modelado de sistemas de control para las intersecciones. A continuación se presenta un modelo representativo en RP de STU.

Modelado con RP Continúas e Hibridas

El modelado y control del comportamiento de tráfico con RP continuas, ha sido muy difundido, principalmente en el uso de RP híbridas, esto es RP que tienen partes con comportamiento de RP continuas y partes con comportamiento de RP discretas. Júlvez y sus colegas **[51]** abordan el problema de la explosión de estados generada por la cantidad de vehículos que se encuentran en la red de tráfico (más de un millón para modelos de gran escala), adoptando un enfoque macroscópico usando variables como la densidad, la velocidad y el flujo. Usan el mismo concepto de transiciones de velocidad variable o semántica de servidores infinitos de Tolba **[48]**, logrando simular con gran exactitud el flujo de tráfico a través de una sección de calle, de esta manera de acuerdo al valor de la densidad en el segmento de calle siguiente, será modificado el valor de la densidad en el segmento de la calle anterior, generándose una congestión a partir de la sección de calle siguiente y todas las que la preceden.

En este modelo se agregan 3 lugares especiales al modelo de un segmento. Cada uno de ellos representa respectivamente la capacidad del carril, el máximo flujo posible y la capacidad de la siguiente sección, como se muestra en la Fig. 1.4:

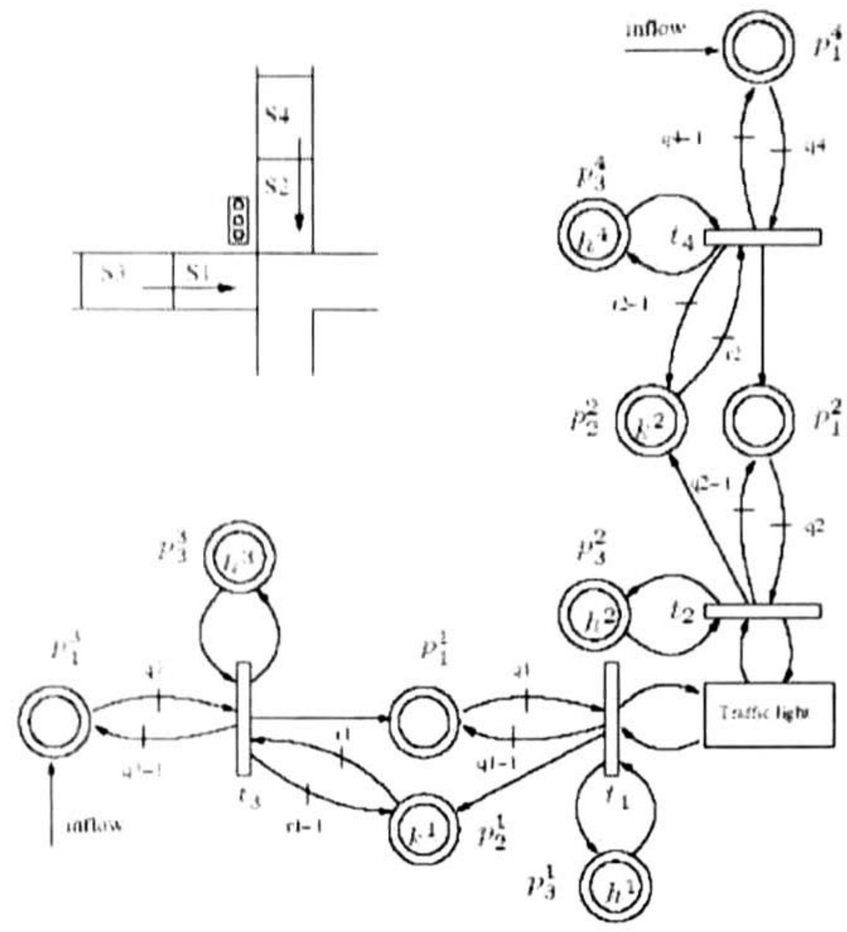


Fig. 1.4: Red de Petri, representando una intersección.

Actualmente la mayoría de los modelos de RP de STU, describen el flujo de tráfico como continuo, en lugar de manera discreta. En parte, por la búsqueda de lograr relajar el sistema y evitar en lo posible, la explosión de estados debida a lo complejo de la red y la gran cantidad de entidades de que está formado el STU. Se observa, por ejemplo, que los parámetros de desempeño como el retardo, el grado de saturación y la longitud de colas son derivadas de funciones analíticas las cuales dependen ya sea de una representación idealizada del sistema o métodos empíricos derivados de datos de campo *(Barclay, 1999)*. Conforme el nivel de diseño de los modelos de tráfico se incrementa, los modelos determinísticos se vuelven más problemáticos. En condiciones de congestión, las funciones retardo-volumen tienden a ser inestables y las interacciones entre las intersecciones y los efectos de segmentos de calle muy pequeños introduce discontinuidades en las funciones. Aunque sea teóricamente posible incrementar el sistema que es analizado y adoptar funciones más complicadas, en práctica, el sistema fácilmente se vuelve inmanejable *(Barclay, 1999)*. Podemos concluir que no es satisfactorio representar al tráfico como flujo continuo ya que por ejemplo, el número de arribos en el periodo de análisis no necesariamente iguala el número de salidas, precisamente causado por la discontinuidad de las funciones en redes muy grandes.

1.2.5. Paradigma multi-agente

Un enfoque utilizado para la simulación de los STU son los SMA, estos se componen de entidades autónomas con objetivos individuales interrelacionados que pueden percibir el ambiente que les rodea y actuar en él de manera inteligente, siguiendo ciertas políticas de comportamiento del ambiente en el que se desenvuelven. El nivel de inteligencia podría variar desde roles y funciones predeterminados hasta una entidad con la capacidad de aprendizaje. Boon define un SMA (ver Fig. 1.5) como un sistema adaptativo complejo y establece que para satisfacer totalmente el requerimiento de un SMA, el SMA necesita tener un ambiente, un grupo de agentes, un conjunto de objetos, y conjunto de operaciones y finalmente algunas leyes [**52**]. Esto es abreviado con la 5-tupla SMA = (*E, A, O, Ops, L*), donde:

- *E* es el ambiente, el lugar donde los agentes residen. Este es un límite que permite definir qué es lo que está dentro y qué es lo que está fuera del sistema a modelar.
- *A* es el conjunto de agentes o entidades autónomas, participando activamente en el sistema. Éstos tienen comportamiento individual y la capacidad para realizar toma de decisiones con el fin de lograr sus objetivos particulares. La acción de los agentes, resultado de la toma de decisiones; pueden incluir alguna de las siguientes actividades: movimiento, percepción y/o modificación del ambiente. Los agentes pueden diferir de otros en sus comportamientos, objetivos, la forma en la que observan el ambiente, diferentes opciones de comportamiento y formas de actuar en el ambiente.
- *O* es el conjunto de objetos, son las partes no autónomas del ambiente. Los objetos pueden ser desde recursos, herramientas, dispositivos, conceptos hasta inclusive categorías. Los objetos son utilizados por los agentes o modificados por los agentes.
- *Ops* es el conjunto de operaciones o procesos que se llevan a cabo en el ambiente. En la mayoría de los casos, las operaciones son parte de los agentes en el sentido que éstos interactúan con el ambiente. Algunas operaciones puede ser más complicadas y residir en múltiples agentes.
- *L* es el conjunto de leyes o Estas son las reglas básicas que gobiernan el ambiente del SMA. Estas leyes son las restricciones que los agentes deben obedecer.

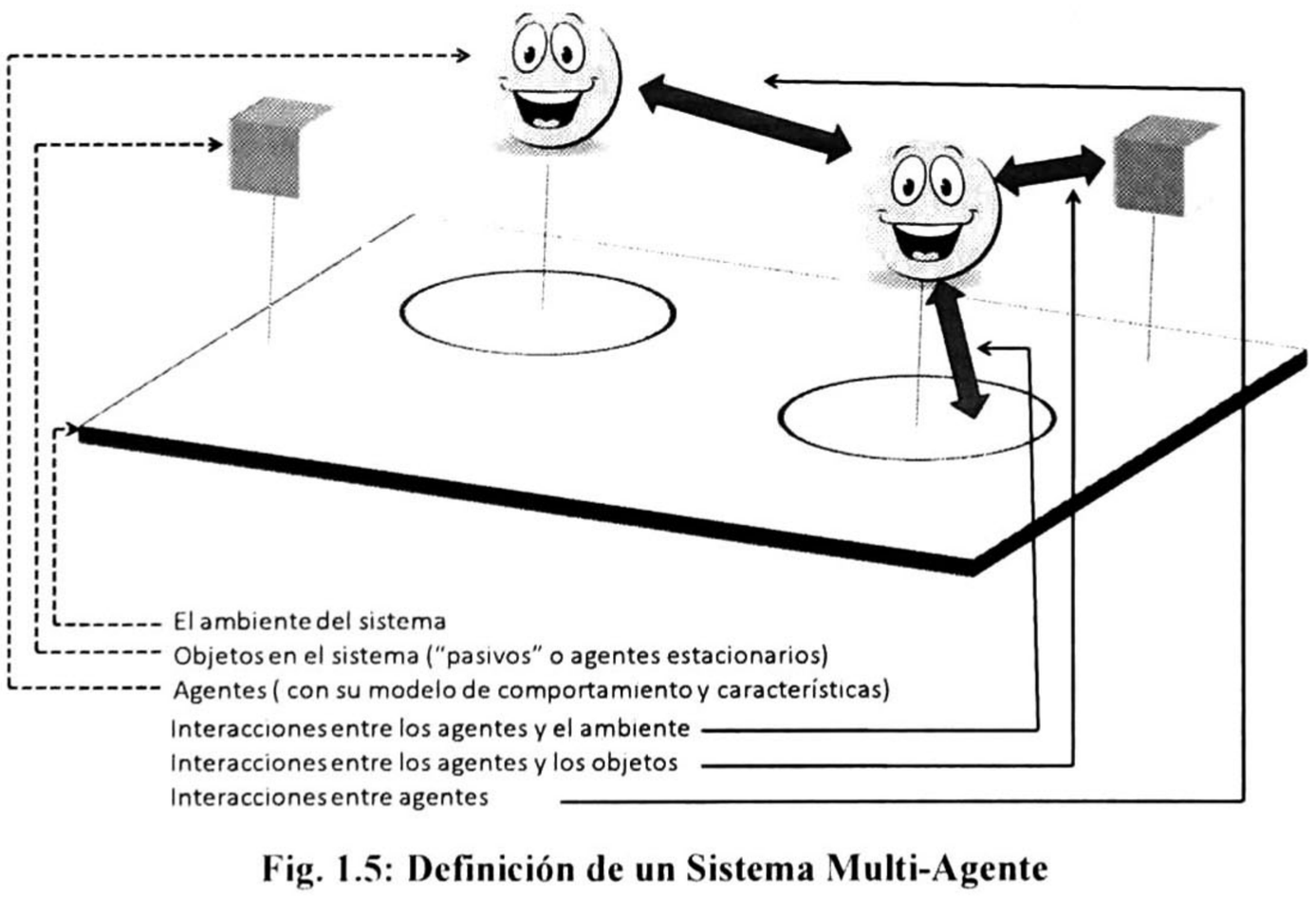


Fig. 1.5: Definición de un Sistema Multi-Agente

El concepto de agente actualmente es usado ampliamente no solo como un modelo de unidades de programación computacional mostrando cierta tipo de características, sino también en una forma más abstracta y general para el análisis, especificación e implementación de sistemas de software complejos . El paradigma SMA ha sido probado como una opción para la construcción de aplicaciones de software adaptativas y complejas, ya que aborda cuestiones clave para hacer que la complejidad se vuelva manejable en el plano conceptual. Además, el paradigma multi-

agente, al enriquecer el mundo de los objetos pasivos con la noción de agentes autónomos; puede ser visto como un sucesor natural del paradigma orientado a objetos. Así pues, el paradigma multi-agente es un área de investigación que se encuentra en rápido desarrollo y tiene el potencial para estimular y contribuir a una amplia variedad de campos científicos [**53**].

Los SMA permiten la simulación del comportamiento dinámico de sistemas complejos a nivel microscópico (como aquellos en los que el ser humano se encuentra involucrado) y el estudio de comportamientos a nivel macroscópico que surgen de la interacción de múltiples agentes [**54**]. Por lo tanto, la aplicación de los SMA en la simulación de STU es adecuada y eficaz.

De acuerdo a la habilidad del agente, éstos pueden ser clasificados en tres tipos: agentes reactivos, agentes cognitivos (deliberativos) y agentes híbridos [**55**][**56**]. Los agentes reactivos toman sus decisiones en tiempo de ejecución, usualmente basados en una cantidad de información muy limitada y reglas simples de tipo situación-acción. Los agentes cognitivos mantienen una representación interna de su ambiente y existe un estado mental explícito que puede ser modificado por algún tipo de razonamiento simbólico. Por último, en algunas situaciones, las habilidades de los dos tipos de agentes son necesarias, tal tipo de agente es un agente híbrido.

MAS-T^{2}er Lab (SMA dirigido por tiempo)

La técnica de avance de simulación de TD es utilizada principalmente en los modelos de SMA, en este sentido el trabajo de Rosetti y sus colegas [**57**][**58**][**14**][**55**] ha sido seleccionado como trabajo representativo de este enfoque. En este trabajo el simulador ejecuta en cada paso de simulación una rutina de temporización que permite al agente tomar decisiones usando el estado de su ambiente y atributos locales. El intervalo de tiempo para la ejecución de un paso de simulación normalmente es de 100 milisegundos.

En cada paso de simulación cada conductor que se encuentra en la red tendrá que decidir la siguiente actividad a desempeñar. Después que el conductor decida la acción a realizar son actualizadas la velocidad y posición del vehículo. La velocidad es actualizada de acuerdo al factor de aceleración del vehículo. La posición es actualizada en base al valor actualizado de la velocidad. La velocidad es calculada en metros por paso de simulación, multiplicada por un valor definido por el usuario (llamado el valor multiplicador). Esto significa que si el usuario desea tener una simulación en tiempo real, deberá definir el valor multiplicador a uno. Si el usuario define 100 milisegundos como el paso de simulación, esto significa que la velocidad del vehículo y la posición será calculada asumiendo que ya han transcurrido 100 milisegundos (por ejemplo, la posición actualizada es igual a la suma de la posición actual con el producto de la velocidad por 0.01 segundos). Si el usuario define un multiplicador de dos, entonces cada 100 milisegundos la velocidad y la posición del vehículo serán calculadas en cada paso de simulación como si ya hubieran transcurrido 200 milisegundos (pero en la realidad han pasado solo 100 milisegundos). El diagrama de estados de la Fig. 1.6 ilustra la dinámica de la simulación.

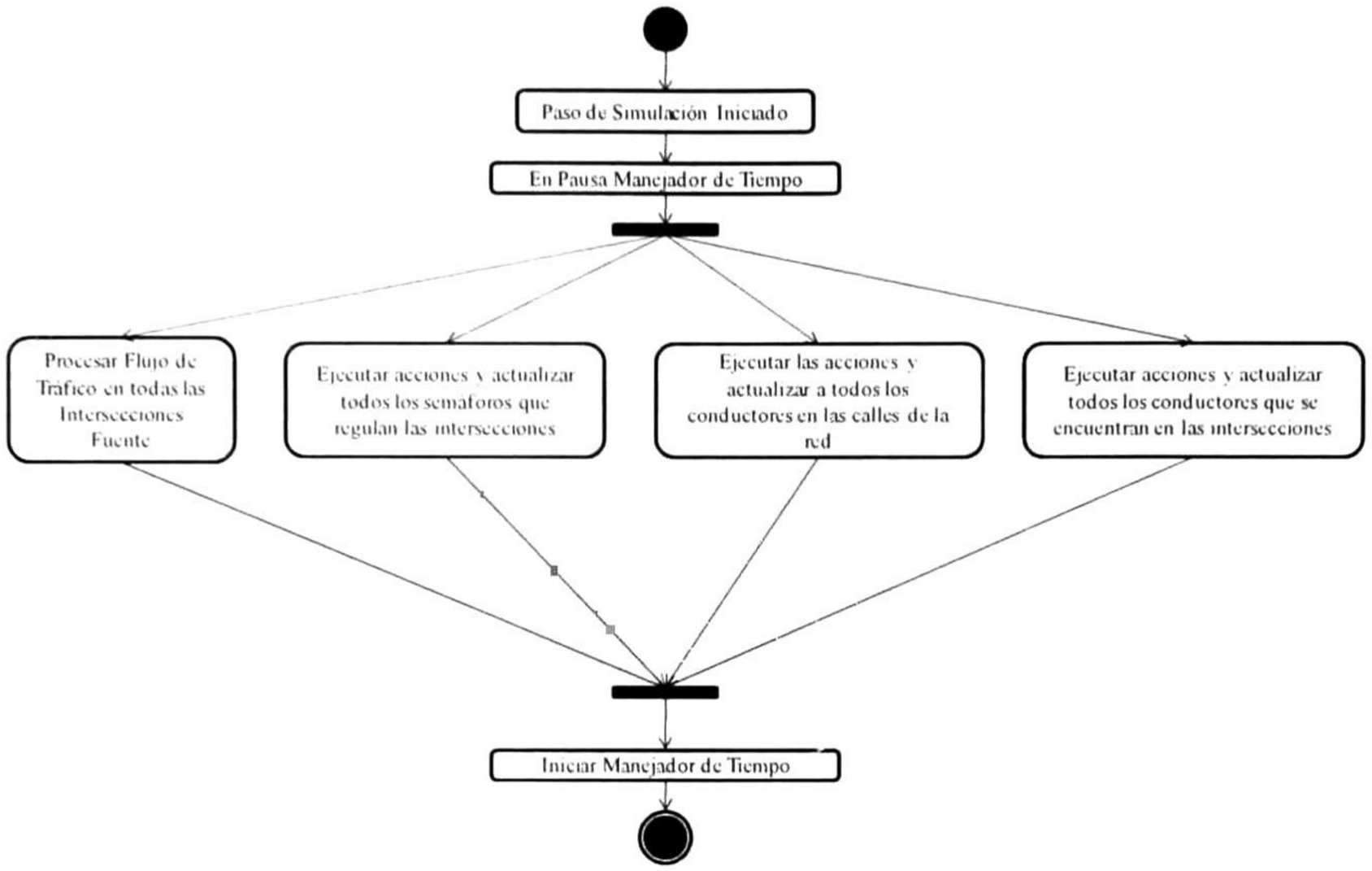


Fig. 1.6: Diagrama de Estados del paso de Simulación en MAS-T^{2}er-Lab[59]

Plataforma de experimentación para SMA y Estrategias de Control de Tráfico Urbano (SMA dirigido por eventos)

La técnica ED, es utilizada principalmente en los SMA, para resolver el problema de control dinámico de tráfico. En este esquema, en lugar de ver el problema como un control centralizado, ha sido visto como un control descentralizado y predictivo. De esta manera los agentes reciben información y toman decisiones para mejorar u optimizar el flujo de la red [**60**][**61**][**62**][**63**][**22**] [**23**][**24**].

En este sentido el trabajo de Katwijk y sus colegas [**60**] ha sido seleccionado como trabajo representativo de este enfoque. En éste se presenta el desarrollo de una cama de pruebas que facilita el desarrollo de SMA para la administración dinámica del tráfico de una red urbana. Divide el proceso de control de tráfico en dos partes. La primera es la simulación del flujo de tráfico vehicular. Este proceso puede ser observado por medio de equipos de monitoreo (e.g. detectores de ciclo inducido, cámaras, sensores, etc.) e influenciado por los instrumentos de control de tráfico (e.g. los señalamientos de mensaje variable, semáforos, instalaciones de medición de acceso a rampas, etc.), los cuales forman el material de trabajo para el segundo proceso. El control del proceso se lleva a cabo por múltiples agentes que interactúan constantemente.

El proceso de simulación de tráfico es realizado en PARAMICS [**64**]; el control del proceso es delegado a un SMA implementado en JADE[**65**]. PARAMICS simula el tráfico a nivel de vehículos individuales (microscópico) y se diseña un agente genérico basado en reglas usando JESS[**66**]. En PARAMICS es posible requerir información del estado de la simulación de los detectores de tráfico simulados en cada paso de simulación. Existe solo un agente que comunica esta información a los demás agentes; cuando alguno de los agentes requiere de más tiempo que el asignado en el paso de simulación, el agente de control suspende la simulación hasta que todos los agentes estén listos para continuar. Los agentes siguen un protocolo de comunicación que permite enviarse mensajes para tomar decisiones. La Fig. 1.7 muestra un ejemplo en el cual la avenida principal tiene tres agentes que constantemente están recibiendo el estado de esta a través de los sensores colocados en la avenida, si alguno decide que el

flujo es muy alto entonces requerirá a alguno de los agentes de las rampas de acceso que disminuya el flujo a la avenida modificando los tiempos de los semáforos ubicados en estas rampas de acceso.

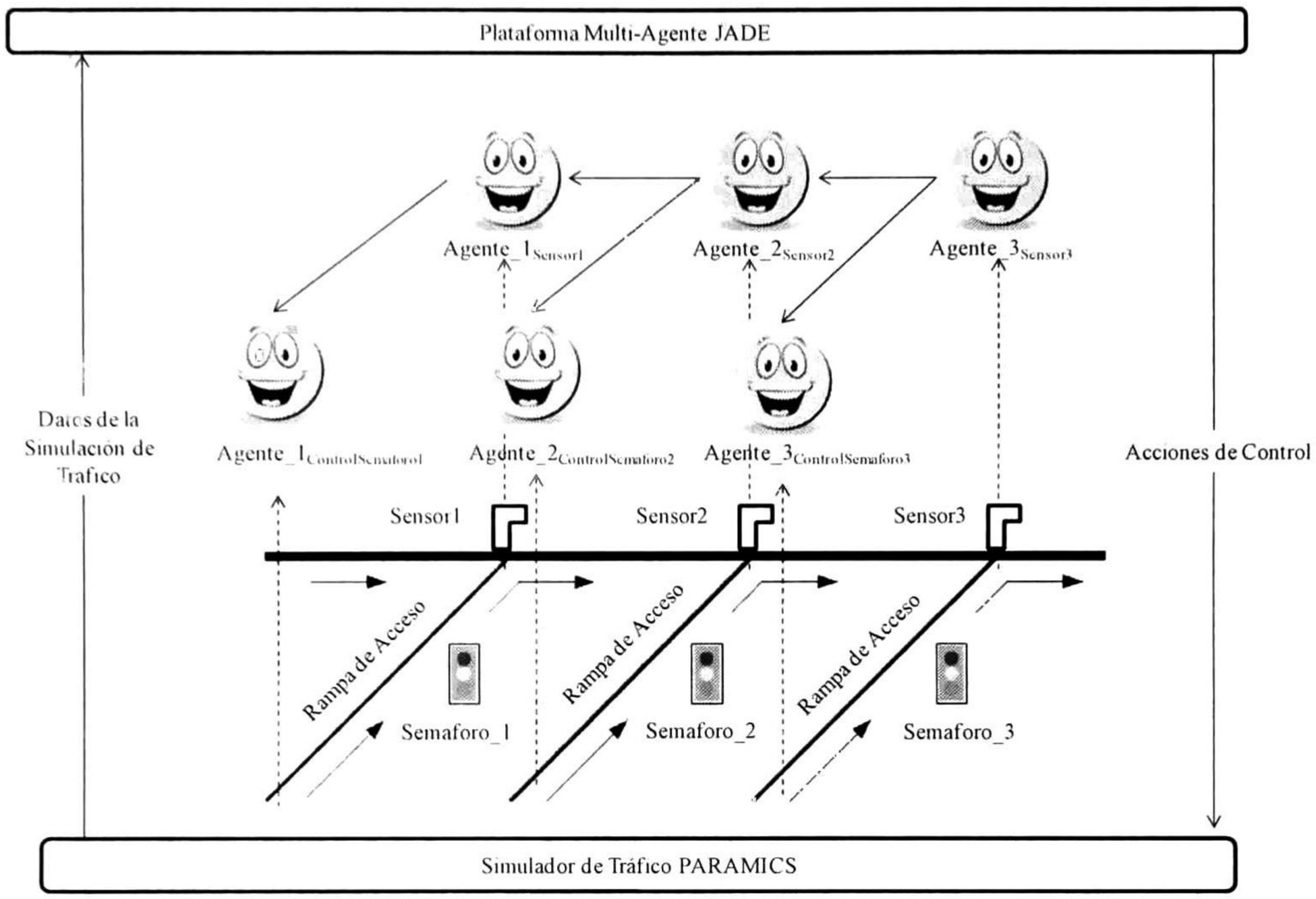


Fig. 1.7: Coordinación de Agentes para la implementación de estrategias de control [60].

1.3. Simulación distribuida de sistemas de tráfico urbano

Para distribuir la ejecución de la simulación de los modelos presentados en la sección anterior, se ha usado la estrategia de descomposición de dominio **[67][68]** y la descomposición funcional o por tareas **[69]**. A cada unidad de procesamiento (nodo) se le asigna un fragmento de área del STU modelado y se sincroniza en cada paso de simulación. Este conjunto de nodos (clúster) están unidos mediante una red de alta velocidad y la sincronización se realiza a través de mensajes entre nodos usando un *middleware*. El middleware es un software de conectividad que permite la comunicación transparente entre diferentes nodos distribuidos a lo largo de la red (Internet, Intranet, etc.), tales como MPI (*Message-Passing-Interface*) o PVM (*Parallel-Virtual-Machine*).

En particular los SMA distribuyen los agentes a través de la red interactuando por algún protocolo de comunicación **[70][71]**. Si el clúster es pequeño (pocos nodos) y la partición es óptima (menor número de mensajes de salida entre nodos) este tipo de distribución provee de un incremento potencial en el valor de *speed-up*. Sin embargo, debido al costo computacional causado por la sincronización, esta solución no es óptima para clúster de gran tamaño **[46]**.

Una estrategia para disminuir el costo computacional debido a la sincronización realizada en cada paso de simulación, es usar la técnica ED para realizar el avance de la simulación. En ésta, el estado del sistema cambia solamente en intervalos irregulares, con un número finito de puntos de ejecución en el tiempo. Cada punto corresponde a la ejecución futura u ocurrencia de un evento. Un evento puede cambiar el estado del

sistema. Charypar y sus colegas [**69**], para superar la ineficiencia del enfoque de teoría de colas, presente en áreas donde la relación flujo-densidad es baja (ya que para cada paso de simulación en cada segmento se verifica si existe algún vehículo listo para dejar el segmento), utilizan la técnica ED representando el instante de salida del vehículo que esté listo para dejar el segmento, con la marca de tiempo de un evento; acelerando hasta en un valor de 20 la relación de velocidad de simulación en paralelo, con respecto a la simulación secuencial hasta en 20 (*speed-up* = 20). De modo que, la distribución de este modelo se deja como trabajo futuro, proponiendo el uso mecanismos de sincronización del enfoque de simulación de eventos discretos paralela y distribuida (SEDP).

La SEDP es un enfoque ampliamente usado para tener una simulación descentralizada y distribuida de modelos ED. En este enfoque, el modelo se divide en un conjunto de procesos lógicos (PL), cada uno manteniendo y procesando una porción de espacio del ambiente del sistema. Los cambios de estado son modelados como eventos.

Podemos clasificar los eventos en dos tipos: los eventos internos los cuales tienen un impacto causal únicamente en las variables de estado de un PL, y los eventos externos que también pueden tener un impacto en las variables de otros PL, los cuales son modelados típicamente como mensajes con marca de tiempo intercambiados entre los PL implicados. Una variable de estado se dice que es compartida, si se lee o es actualizada por eventos externos generados por más de un PL.

Cada PL mantiene un reloj local con el valor actual del tiempo de simulación, es decir, la hora local virtual (HLV). Este valor representa la vista local del proceso con respecto al tiempo global de simulación y denota hasta qué punto en el tiempo el proceso correspondiente ha avanzado. La marca de tiempo de un mensaje es el HLV del PL fuente cuando el mensaje fue enviado.

Una cuestión fundamental en la simulación paralela y distribuida es la sincronización de los PL para asegurarse que los mensajes siempre se procesen en orden no-decreciente de acuerdo a la marca de tiempo de los mensajes. De esta manera se conservan las dependencias causales y el orden parcial de los eventos del sistema. En el enfoque SEDP síncrono se utilizan mecanismos de sincronización globales para obligar a los PL a hacer avanzar el mismo tiempo de simulación, y así mantenerlos perfectamente sincronizados. En cambio, en los SEDP asíncronos, las HLV de cada PL avanzan a diferentes tiempos de simulación, procesando concurrentemente eventos que ocurren en diferente tiempo de simulación y diferentes PL.

Aunque éste último enfoque tiene una mayor *speed-up*, se requieren mecanismos adicionales para garantizar que cada PL mantenga una relación de causalidad local válida procesando los eventos en orden no decreciente de acuerdo a su marca de tiempo (*look-back*, *rollback*, etc.) [**72**][**73**][**74**]; dependiendo del tipo de enfoque utilizado se pueden clasificar como mecanismos optimistas [**75**] [**76**] y conservadores [**77**] [**78**].

Esta estrategia es adecuada para el modelado de sistemas con inherente paralelismo asíncrono, tal como los SMA. En trabajos anteriores [**79**][**80**][**81**][**82**][**83**] se han propuesto una aproximación a la simulación de un agente basado en sistemas basados en el paradigma de PL [**84**], en la que los agentes y su entorno son modelados como un conjunto de PL. Este enfoque modela cada agente como un simple *Proceso Lógico de Agente* (PLA). Del mismo modo, suponemos que los objetos y procesos dentro del ambiente de los agentes son modelados como uno o más procesos lógicos de ambiente (PLE). Los PLA interactúan con los PLE y otros PLA vía eventos externos. La finalidad de esta interacción es el intercambio de información relativa a los valores de las variables del estado compartido, las cuales definen el ambiente del agente y las interfaces entre los PLE. Un problema clave es cómo gestionar el intercambio de

mensajes externos entre los PLA y PLE para garantizar la escalabilidad de la simulación.

El enfoque de división por PL en los SEDP restringe la capacidad de ejecución concurrente dentro de estos PL, al ser descritos como la unidad básica de modelado, no permiten el desarrollo de algoritmos que obtengan ventaja de la naturaleza concurrente de los sistemas complejos, tales como el STU. Además, la mayoría de los trabajos se han enfocado en la optimización de algoritmos para acelerar los tiempos de simulación; pero se han ignorado las metodologías que permitan integrar diferentes estrategias, que permiten la no-limitación en la capacidad del nivel de detalle del modelo.

1.4. Discusión y propuesta de tesis

La Tabla 1.1 muestra la clasificación de los modelos representativos realizada en este capítulo.

Trabajo Relacionado	Dirigidos por Tiempo Discreto	Dirigidos por Eventos Discretos	Técnica de Modelado
TRANSIMS (Nagel, et. al., 1997)	✔		Autómatas Celulares
SCATTER (Perumalla,et. al., 2006)	✔	≈ (En Semáforos)	Procesos Lógicos
AIMSUN (Barceló, et. al., 2002)	✔		Vehículo-Siguiente
MATSIM-T (Cetin, et. al., 2003)	✔		Teoría de Colas
MAS-T^{2}er Lab (Rosetti, et. al., 2009)	✔		Sistemas Multi-agente
MATSIM-T (Nagel, et. al., 2007)		✔	Teoría de Colas
(Katwijk, et. al., 2005)	✔	≈ (En Semáforos)	Sistemas Multi-agente

Tabla 1.1:Modelos de Simulación de Tráfico Representativos

En esta sección analizaremos las ventajas y desventajas (ver Tabla 1.2) de los diferentes enfoques presentados previamente. Además, haremos un resumen de los retos encontrados en estos enfoques para el modelado y simulación de STU.

Trabajo o estudio	Ventajas	Desventajas
Modelos vehículo siguiente	• Descripción detallada del avance del vehículo. • Algunos de estos modelos permiten incorporar y evaluar STI [**85**][**86**].	• Son diseñados para algún tipo específico *(ad-hoc)* de STI y no permite la incorporación o diseño de nuevas medidas STI en el modelo. • Se ha probado que una fuente de error en la simulaciones se debe a la longitud arbitraria del paso de simulación

		[19][20].
Autómatas Celulares	• Usando reglas básicas se pueden generar y observar comportamientos complejos • Llevarlo a la implementación es muy sencillo. • Provee de una imagen física simple del sistema y puede ser modificado fácilmente para evaluar diferentes aspectos del STU.	• Los AC no son la mejor opción para describir el movimiento de entidades ya que la actualización síncrona no es la adecuada para generar resultados realistas. • Aquellos modelos que lo utilizan **[87][88]** no describen AC canónicos **[89]** • No obstante, cuando se quiere realizar simulaciones con estos modelos a gran escala, no resultan ser lo más óptimos, debido al alto costo computacional causado por la actualización de los modelos de las tareas de manejo de cada vehículo en cada paso de simulación y a la reducción de velocidad computacional del modelo **[21]**. • La resolución espacial y temporal de los CA es muy baja..
Teoría de Colas	• Tiempos de Simulación Eficientes	• Pérdida de resolución espacial en los segmentos, no apto para simulación de STI en STU.
Procesos Lógicos	• Encapsulamiento Proceso • Algoritmos de Ejecución Distribuida	• Fuerte Relación de Causalidad en cada proceso lógico
Multi-Agentes	• Facilita bibliotecas para la rápida implementación de simulaciones, es microscópico y permite evaluar inteligencia.	• La mayoría de ellos son sistemas ad-hoc, lo cual hace complicado el agregar algún nuevo componente al modelo o integrar medidas STI. • La mayoría de las plataformas SMA siguen un paradigma fundamentado en el uso de bibliotecas, al usar este paradigma, un concepto de diseño y metodologías apropiadas para el diseño de nuevos componentes está ausente.

Tabla 1.2: Ventajas y Desventajas de los Métodos de Modelado de STU

La evaluación de dispositivos de control como los STI, requieren de una alta resolución espacial y capacidades de razonamiento de los vehículos para que las simulaciones sean realistas. Por otro lado, el enfoque multi-agente de ED parece ser el más prometedor, sin embargo, la mayoría de las plataformas de SMA operan bajo la técnica de TD y con el uso de bibliotecas, donde un lenguaje de programación huésped es extendido a través de procedimientos genéricos para facilitar el desarrollo de SMA. En este enfoque, la biblioteca incluye un marco de desarrollo (un conjunto de conceptos para el diseño de SMA) junto con las herramientas para la simulación y visualización.

Aunque, al usar este paradigma, las metodologías apropiadas para el diseño de los SMA dirigidos por eventos están ausentes.

Uno de los principales problemas de los simuladores existentes, es la falta de metodologías y conceptos de diseño que permitan integrar nuevos componentes a los modelos propuestos [**90**][**53**], tal como es requerido en los STU actuales que usan dispositivos STI en sus sistemas de control. Chabrol y sus colegas proponen una metodología que permite el uso del concepto de SMA utilizando UML [**91**] y se ha intentado relacionar la información entre el nivel físico del sistema con el dinámico [**92**]. Aunque, son metodologías que utilizan métodos no formales, limitando la capacidad de análisis del modelo, capacidad requerida en modelos de SMA, dada su complejidad y cantidad de variables de estado. La simulación de los SMA requiere de una gran cantidad de recursos computacionales, así pues, se han propuesto estrategias en las que el ambiente se representa como una cuadrícula; una colección de entidades móviles y un conjunto de reglas para actualizar ambos componentes.

De los modelos estudiados y que han abordado la integración de los SMA dirigido por eventos, podemos establecer tres características requeridas del concepto de diseño para este tipo de modelos:

- Debe ser flexible, es decir, no estar confinado a estructuras exclusivas de SMA o la técnica ED.
- Debe soportar modularidad y reusabilidad para poder beneficiar investigaciones futuras.
- No debe eliminar alguno de los beneficios, dados por técnica ED (simplicidad y manejo de eventos eficientes) y SMA (modelado de comportamiento complejos).

Además, se requieren de herramientas de diseño simples y robustas que soporten ambos enfoques.

La principal ventaja del uso de la técnica ED en el diseño de la arquitectura de los simuladores, con respecto a la TD, es la eliminación de los intervalos en los que no ocurren cambios en el estado de los componentes del sistema, permitiendo capturar sólo aquellos puntos de decisión que son de interés para el modelo y que son capturados como eventos en el modelo.

Si bien, aunque se ha probado la eficiencia de la técnica ED para el modelado de sistemas complejos [**4**][**19**], no se tiene conocimiento de algún trabajo que utilice el técnica ED para la descripción del STU; particularmente en el avance del vehículo en una calle a nivel microscópico. La mayoría de los modelos existentes utilizan la técnica TD, esto es: los modelos de entrada de los vehículos (*car-following, gap-search, lane-change*, etc.) son actualizados cada segundo o menos.

Generalmente la técnica ED es utilizada únicamente para describir los comportamientos de los vehículos en las intersecciones o el cambio de estado de los semáforos [**4**]; así como para describir la llegada y salida de un vehículo en una calle utilizando eventos [**11**].

La ausencia de una metodología que permita capturar el comportamiento de las componentes dinámicas del STU utilizando la técnica ED, la cual no es tan intuitiva como la TD, puede ser la causa de no contar con el nivel de detalle microscópico necesario para el modelado de las aplicaciones de tráfico inteligentes utilizadas en los STU actuales.

Se requieren modelos que soporten características de modelado avanzado tales como: modelado modular y jerárquico, simulación distribuida, capacidades de abstracción del modelo. Las ventajas de estas capacidades como la reducción en el

tiempo de desarrollo del modelo, soporte en la reutilización de una base de datos de modelos y ayuda en la verificación y validación del modelo son bien aceptados. En estudios ITS especialmente, estas capacidades proveen de medios convenientes para construir y analizar sistemas de red de tráfico complejos.

El enfoque de división por PL en los SEDP, restringe la capacidad de ejecución concurrente dentro de estos PL. Al ser descritos como la unidad básica de modelado no permite el desarrollo de algoritmos que obtengan ventaja de la naturaleza concurrente de los sistemas complejos, tales como el STU. Además, la mayoría de los trabajos se han enfocado en la optimización de algoritmos para acelerar los tiempos de simulación, pero se ha puesto de lado las metodologías que permitan integrar diferentes estrategias, sin necesidad de limitar la capacidad del nivel de detalle del modelo.

En este trabajo se utiliza la metodología de modelado microscópico y jerárquico de un STU utilizando el enfoque de SMA dirigido por eventos y se proponen algunas técnicas de ejecución concurrente y distribuida del modelo. Además, se utiliza un enfoque distinto al de otros autores, al dividir la simulación en espacio y tiempo.

Para el modelado de STU usando la integración de SED y SMA, es usado el formalismo de redes de Petri multi-nivel 3-LNS. En el siguiente capítulo describe dicho formalismo.

CAPITULO 2

MODELADO DE SISTEMAS DE TRÁFICO URBANO BASADO EN AGENTES

Este capítulo presenta una metodología para modelado de sistemas de tráfico urbano basado en agentes orientado a eventos utilizando el formalismo de redes de Petri multi-nivel n-LNS. Además, una arquitectura de agente cognitivo basado en reglas es presentada, por último se describe una metodología para capturar la base de conocimiento del agente.

2.1. Herramientas de modelado de sistemas multi-agentes

En el capítulo anterior se puede observar que el uso del paradigma de SMA para el modelado de STU es adecuado y eficaz. Sin embargo, se requiere de un método formal para el modelado de SMA que ofrezca ventajas de representabilidad de la mayoría de sus características como la movilidad, la interacción y la definición del ambiente, así como un acercamiento a la implementación.

De los formalismos que ofrecen representabilidad para SMA se pueden mencionar a la lógica modal **[93][94]**, el álgebra de procesos, lenguajes formales, AUML (*Agent Unified Modeling Language*) **[95]**, los autómatas finitos y las redes de Petri (RP), los tres primeros, entre otras desventajas, distan mucho de la implementación. Este es el caso de los lenguajes declarativos, como la lógica modal, que expresan posibilidades, necesidades, creencias, conocimiento, progresión temporal, incluso movilidad, pero no hay una aproximación a la implementación **[93]**.

Las RP debido a que implican una representación gráfica, ofrecen un alto nivel de representabilidad visual tanto del ambiente como de los agentes, su interacción y movilidad. Aparte de ser un método formal que brinda gran capacidad de análisis de propiedades y de simulación, nos acerca a una propuesta de implementación sin estar atados a ninguna arquitectura de SMA en especial.

En particular nos referimos a las RP multi-nivel, introducidas por primera vez por Valk **[96]**, con el concepto de "redes dentro de redes", basándose en la propuesta de los colores de Jensen **[97]** pero ahora utilizando una red como un "color"

H. Almeyda retomó la idea de Valk con su propuesta de "Un sistema de red a tres niveles para el modelado de agentes móviles" **[98]**. En ese trabajo se presenta una extensión a los formalismos de Valk e Hiraishi **[99]**, resultando en un sistema de red de Petri a tres niveles para el modelado de agentes móviles. Se propone una arquitectura de software para la implementación de los modelos obtenidos, usando Java™ como lenguaje de implementación.

N. Villanueva **[100]** hace una generalización de la especificación de H.Almeyda a n-niveles (n-LNS), pero ella lo enfoca a sistemas de procesamiento por lotes; más tarde R. Sanchéz **[101]** lo retoma y hace algunas adecuaciones de tal forma que queda completa la especificación n-LNS, este último lo enfoca a protocolos de interacción en sistemas de agentes móviles.

La especificación n-LNS (ver Apéndice A) nos brinda los aspectos que buscamos: una representación intuitiva y compacta de los SMA, la posibilidad de concurrencia y distribución representando el ambiente como lugares de una RP, los agentes como redes dentro de esos lugares y los comportamientos de los agentes como redes dentro de los lugares de éstos; se pueden expresar la interacción entre agentes y la movilidad, sincronizando el disparo de transiciones entre redes. Además, en este trabajo de tesis, se asocia un mecanismo de razonamiento a las redes que describen el comportamiento de los agentes, para que éstos puedan tomar decisiones.

2.2. Componentes del modelo enriquecido n-LNS de un sistema de tráfico urbano

Usando la definición de STU presentada en el capítulo 1, se presenta a continuación una metodología de modelado de STU utilizando el formalismo de RP multi-nivel, n-LNS.

2.2.1. Arquitectura del Modelo

El modelo n-LNS del STU se compone de tres niveles de abstracción (Fig. 2.1); en el primer nivel se describe el ambiente en el que se desplazan las entidades, esto es, la estructura de red vial (e.g. calles, avenidas, viaductos, desniveles, etc.); en el segundo nivel se representan las entidades que pueden desplazarse, interactuar con otras entidades o ejecutar un proceso de toma de decisión (e.g. peatones, vehículos, semáforos inteligentes, etc.); por último, en el tercer nivel se describe el comportamiento de cada clase de entidad presente en el STU.

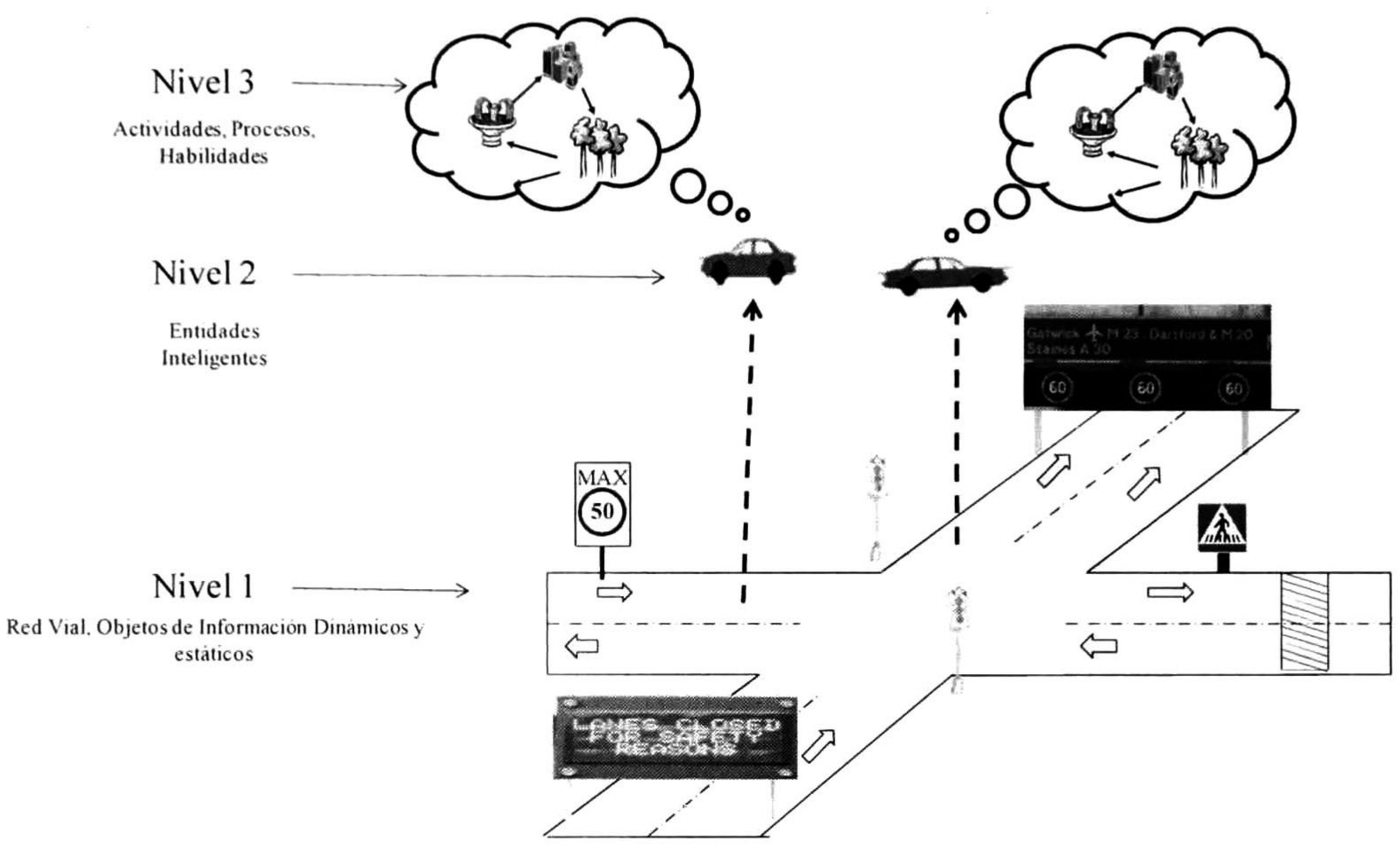


Fig. 2.1: Clasificación del Sistema de Tráfico Urbano con n-LNS

2.2.2. Modelo del ambiente

Red *EnvironmentNET*$_I$

El segmento es la unidad básica de modelado del ambiente del STU (ver Fig. 2.2). Cada segmento representa una vía de la estructura de red vial por la que una entidad puede desplazarse de forma secuencial. Durante su desplazamiento puede encontrar obstáculos de tipo físicos (e.g. topes, boyas, pozos, alguna otra entidad, etc.) o causados al obedecer ciertas políticas de tránsito (e.g. cruce de peatones, zona escolar, semáforos, etc.); entre otros objetos informativos (e.g. señalamientos de ubicación, tableros de información, etc.).

Las entidades contenidas en un segmento deberán desplazarse secuencialmente (en cualquier sentido de dirección); no se permiten desplazamientos contrarios al secuencial. Así, cada uno de los segmentos que describen una intersección del STU solo puede contener un vehículo a la vez (ver Fig. 2.2). Además, el uso de algunos segmentos se restringe a cierta clase de entidades.

La red de nivel 1 ($EnvironmentNET_1$) del modelo n-LNS que representa el ambiente donde se desplazan las entidades del STU, se obtiene de la descripción física de la estructura de red vial; cada lugar de la red $EnvironmentNET_1$ representa un segmento de red y las transiciones representan la conexión física entre segmentos. A cada lugar de la red $EnvironmentNet_1$ se asocia la descripción de su correspondiente segmento.

Definimos el conjunto de descripciones de segmentos con $S = \{s_i \mid s_i$ es la descripción de un segmento$\}$. A continuación se presenta la definición de un objeto en el STU y la de descripción de segmento.

Definición 2.8. Un objeto $\boldsymbol{obj_i}$ es un 3-tupla definida por $\boldsymbol{obj_i} = (type_i, value_i, w_i)$ donde:

- $type_i \in \{señalamientoEstático, signoMensajeVariable, tope, boya, semáforo, signoCedaPaso, signoParoTotal, signoCanalizaciónIntersecciones\}$.
- $value_i$ es la información que se provee a la entidad, tal que $value_i \in \aleph$.
- w_i es la posición relativa del objeto en el segmento, tal que $w_i \in \Re^+$

Definición 2.9. Un segmento $\boldsymbol{s_i}$ es un 4-tupla definida por $\boldsymbol{s_i} = (O_i, typeS_i, a_i, b_i)$ donde:

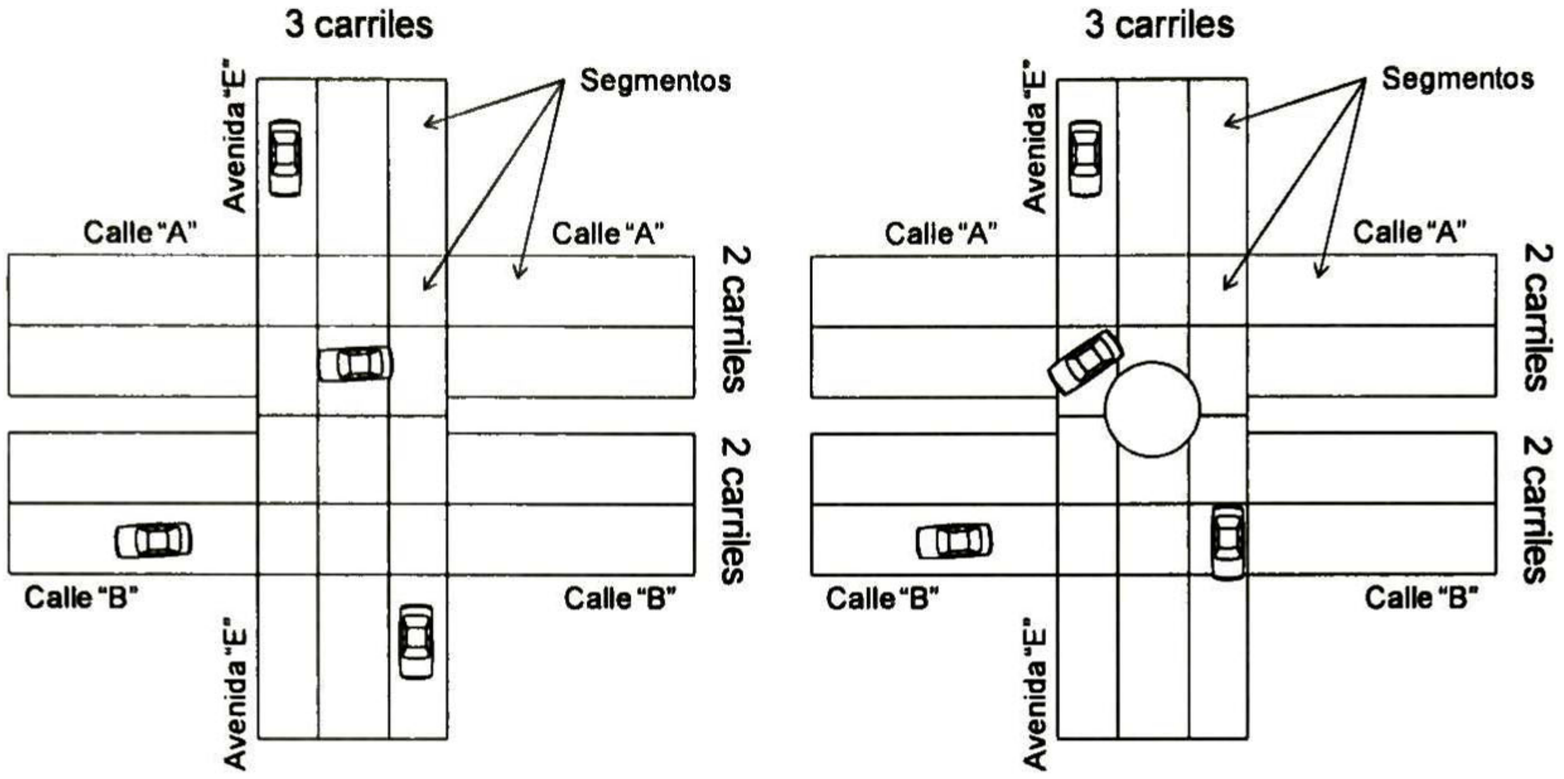
- $O_i = \{obj_j\}$ es el conjunto de objetos en el segmento i.
- $\mathbf{typeS_i} \in \{\mathbf{uso_{vehicular}}, \mathbf{paso_{peatonal}}, \mathbf{exclusivo_{autobus}}, \mathbf{exclusivo_{tren}}\}$ representa el tipo de segmento.
- $a_i, b_i = (lat_i, long_i)$ y $lat_i, long_i \in \Re$, los cuales son las coordenadas geométricas (latitud y altitud) que describen la posición en un mapa de los puntos extremos del segmento.

Las siguientes relaciones permiten establecer el tipo de conexión entre segmentos:

Relación 2.1: Vecindad Secuencial $NS = \{(s_i, s_j) \mid s_i s_j \in S$, las entidades pueden desplazarse secuencialmente del segmento s_i al segmento s_j, agregándose a la cola del extremo s_j por el que se encuentran conectados$\}$. Si $\exists(s_i, s_j) \epsilon NS \rightarrow \exists(s_j, s_i) \epsilon NS$, ya que lo que describe la relación NS es la conexión física posible entre dos segmentos.

Si se modela la habilidad de la entidad para realizar un cambio de un segmento a otro de forma transversal (cambio de carril) se agrega la siguiente relación:

Relación 2.2: Vecindad Contigua $NC = \{(s_i, s_j) \mid s_i s_j \in S$, las entidades pueden desplazarse transversalmente del segmento s_i al segmento s_j en cualquier posición de dicho segmento$\}$. Si $\exists(s_i, s_j) \epsilon NC \rightarrow \exists(s_j, s_i) \epsilon NC$, ya que lo que describe la relación NC es la conexión física posible entre dos segmentos.



a) Intersección de dos calles y una avenida b) Glorieta de dos calles y una avenida

Fig. 2.2: El segmento como unidad de modelado básica

Dado que el sistema modelado es delimitado a un área específica, podemos definir a los segmentos que están ubicados en los bordes del sistema como segmentos fuente y pozo; los cuales representan la entrada y salida de las entidades del sistema respectivamente. Se definen los siguientes predicados:

Sea *fuente* un predicado en S,

$$fuente: S \rightarrow \{true, false\}$$

Entonces,

$$\forall s_i \in S \left(fuente(s_i)\right) \leftrightarrow \nexists s_j \in S \left((s_j, s_i) \in NS\right)$$

Sea *pozo* un predicado en S,

$$pozo: S \rightarrow \{true, false\}$$

Entonces,

$$\forall s_i \in S \left(pozo(s_i)\right) \leftrightarrow \nexists s_j \in S((s_i, s_j) \in NS)$$

Para construir la RP que describe la estructura de red vial, se aplica el siguiente procedimiento:

- Se genera una estructura de RP G, asignando para cada $s_i \in S$ un lugar $p_i \in P$, tal que P describe los lugares de la red G.
- Para cada par $(s_i, s_j) \epsilon NS$ o $(s_i, s_j) \epsilon NC$, se genera una transición $t_{ij} \in T$, junto con los arcos (p_i, t_{ij}), (t_{ij}, p_j) tal que T describe las transiciones de la red G.
- Se agrega una transición t_i por cada segmento s_i fuente o pozo; son agregados respectivamente arcos (t_i, p_i) o (p_i, t_i).

Se construye una red $EnvironmentNET_1$= ($TypeEnvironmentNET$, μ_{en}), con G como su estructura de RP, y para el caso de la Fig. 2.3, $\mu_{en}(p_{11}) = \mu_{en}(p_5) = \{EntityNET_{2,1}\}$ como marcado inicial, donde la red $EntityNET_{2,1}$ representan a los vehículos. Estas redes se describirán más adelante.

El desplazamiento de las entidades ocurre con el disparo de las transiciones; las etiquetas e inscripciones de los arcos contienen únicamente los eventos que permiten a una entidad dejar un segmento y llegar a otro de forma secuencial (*initLeaveSegment*), o el cambio de segmento de forma transversal (*initChangeLane*). Por ejemplo, la transición $t_{0,1}$ con la etiqueta *initChangeLane* permite que una entidad se desplace del lugar p_0 al lugar p_1, entonces la función $\pi((p_0,t_{0,1}), initChangeLane^{\downarrow})$ será agregada. En la Fig. 2.3, se muestra la red de nivel 1 obtenida para el caso de la intersección simple incluida en la misma figura.

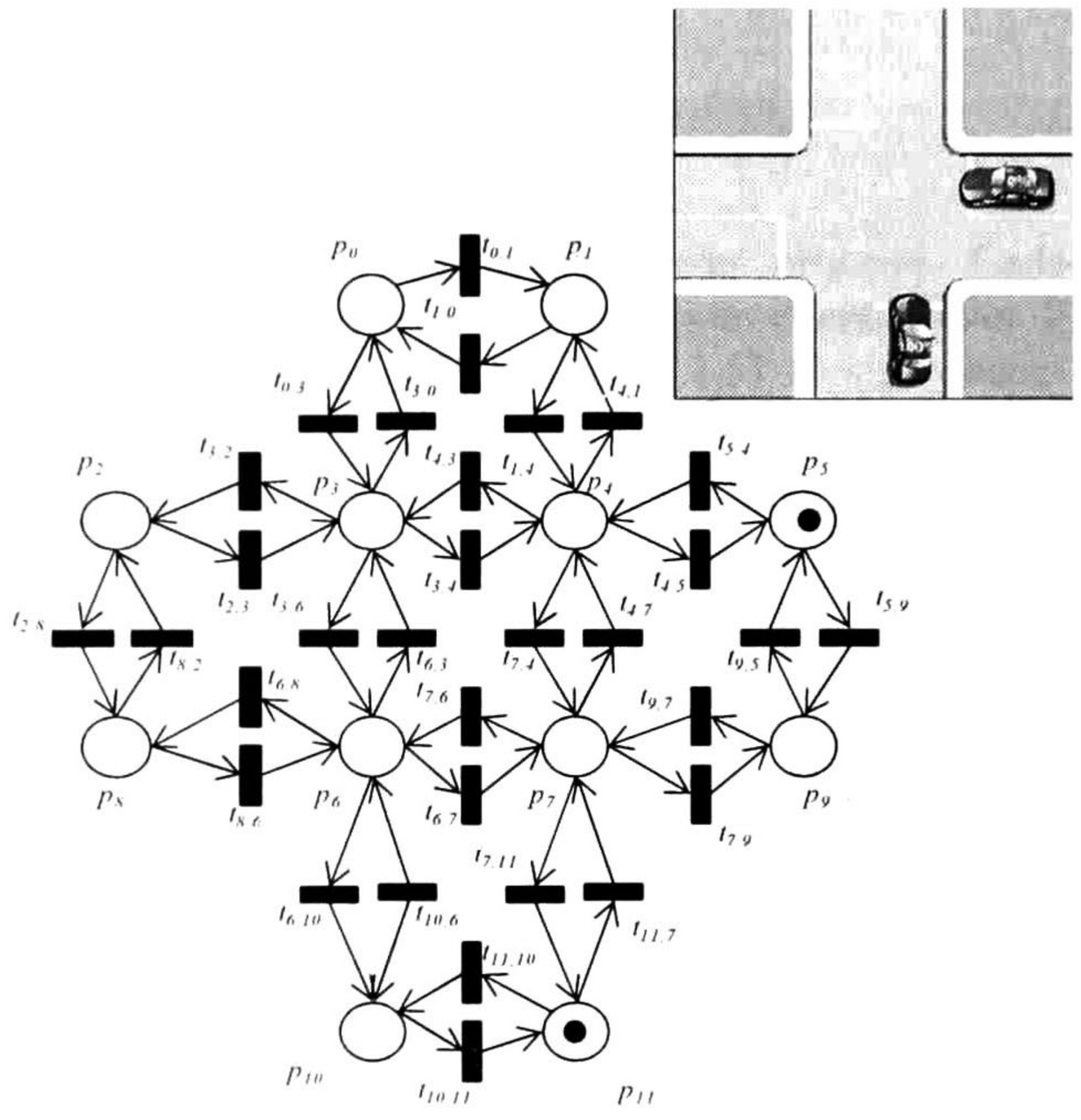


Fig. 2.3: Representación de la estructura de tráfico urbano $EnvironmentNET_1$ en el modelo n-LNS del STU.

Todas estas transiciones deben sincronizarse internamente con las redes nivel 2 de tipo $typeEntityNET_{2,1}$ que marcan sus lugares de entrada. La Tabla 2.1 Tabla 2.1:muestra información adicional para esta red.

TypeEnvironmentNET
$TOKEN_1$ = { $EntityNET_{2,1}$}
$LABEL_1$ = {*initLeaveSegment*, *initChangeLane*}
$VAR_{1,1}$ = {**x** : $TypeEntityNET_{2,1}$ }
λ
$\lambda(t_{0,1})=\lambda(t_{1,0})=\lambda(t_{2,8})=\lambda(t_{8,2})=\lambda(t_{10,11})=\lambda(t_{11,10})=\lambda(t_{3,6})=\lambda(t_{6,3})=\lambda(t_{7,4})=\lambda(t_{4,7})=\{(initChangeLane^{\downarrow})\}$, $\lambda(t_{2,3})=\lambda(t_{3,2})=\lambda(t_{6,8})=\lambda(t_{8,6})=\lambda(t_{6,10})=\lambda(t_{10,6})=\lambda(t_{7,11})=\lambda(t_{11,7})=\lambda(t_{7,9})=\lambda(t_{9,7})=\lambda(t_{5,4})=\lambda(t_{4,5})=\lambda(t_{1,4})=\lambda(t_{4,1})=\lambda(t_{0,3})=\lambda(t_{3,0})=\lambda(t_{3,4})=\lambda(t_{4,3})=\lambda(t_{7,6})=\lambda(t_{6,7})=\{(initLeaveSegment^{\downarrow})\}$

τ
$\tau(p_1) = \tau(p_2) = ... = \tau(p_n) = \{TypeEntityNET_{2,1}\}$, $n = \|S\|$
π
$\pi((p_0,t_{0,1}),initChangeLane^{\downarrow}) = \pi((p_1,t_{1,0}),initChangeLane^{\downarrow}) = \pi((p_2,t_{2,8}),initChangeLane^{\downarrow}) =$ $\pi((p_8,t_{8,2}),initChangeLane^{\downarrow}) = \pi((p_9,t_{9,5}),initChangeLane^{\downarrow}) = \pi((p_5,t_{5,9}),initChangeLane^{\downarrow}) =$ $\pi((p_{10},t_{10,11}),initChangeLane^{\downarrow}) = \pi((p_{11},t_{11,10}),initChangeLane^{\downarrow}) = \pi((p_0,t_{0,3}),initLeaveSegment^{\downarrow}) =$ $\pi((p_3,t_{3,0}),initLeaveSegment^{\downarrow}) = \pi((p_2,t_{2,3}),initLeaveSegment^{\downarrow}) = \pi((p_3,t_{3,2}),initLeaveSegment^{\downarrow}) =$ $\pi((p_3,t_{3,6}),initLeaveSegment^{\downarrow}) = \pi((p_6,t_{6,3}),initLeaveSegment^{\downarrow}) = \pi((p_4,t_{4,3}),initLeaveSegment^{\downarrow}) =$ $\pi((p_3,t_{3,4}),initLeaveSegment^{\downarrow}) = \pi((p_1,t_{1,4}),initLeaveSegment^{\downarrow}) = \pi((p_4,t_{4,1}),initLeaveSegment^{\downarrow}) =$ $\pi((p_5,t_{5,4}),initLeaveSegment^{\downarrow}) = \pi((p_4,t_{4,5}),initLeaveSegment^{\downarrow}) = \pi((p_4,t_{4,7}),initLeaveSegment^{\downarrow}) =$ $\pi((p_7,t_{7,4}),initLeaveSegment^{\downarrow}) = \pi((p_7,t_{7,9}),initLeaveSegment^{\downarrow}) = \pi((p_9,t_{9,7}),initLeaveSegment^{\downarrow}) =$ $\pi((p_7,t_{7,11}),initLeaveSegment^{\downarrow}) = \pi((p_{11},t_{11,7}),initLeaveSegment^{\downarrow}) = \pi((p_6,t_{6,10}),initLeaveSegment^{\downarrow}) =$ $\pi((p_{10},t_{10,6}),initLeaveSegment^{\downarrow}) = \pi((p_6,t_{6,8}),initLeaveSegment^{\downarrow}) = \pi((p_8,t_{8,6}),initLeaveSegment^{\downarrow}) = \{\mathbf{x}\}$

Tabla 2.1: Funciones λ,τ and π de la red *EnvironmentNET*$_1$

2.2.3. Modelo de las entidades

Red *EntityNET*$_2$

Una entidad puede encontrarse ejecutando algún proceso. Cada proceso está compuesto por una secuencia de actividades iniciadas por eventos (ver Fig. 2.4). Este enfoque es capturado por la red *EntityNET*$_2$ (ver Fig. 2.5) al sincronizar las redes tipo *ProcessNET*$_3$, *ActivitiyNET*$_3$ y *SkillNET*$_3$, (éstas serán descritas con detalle en una sección posterior).

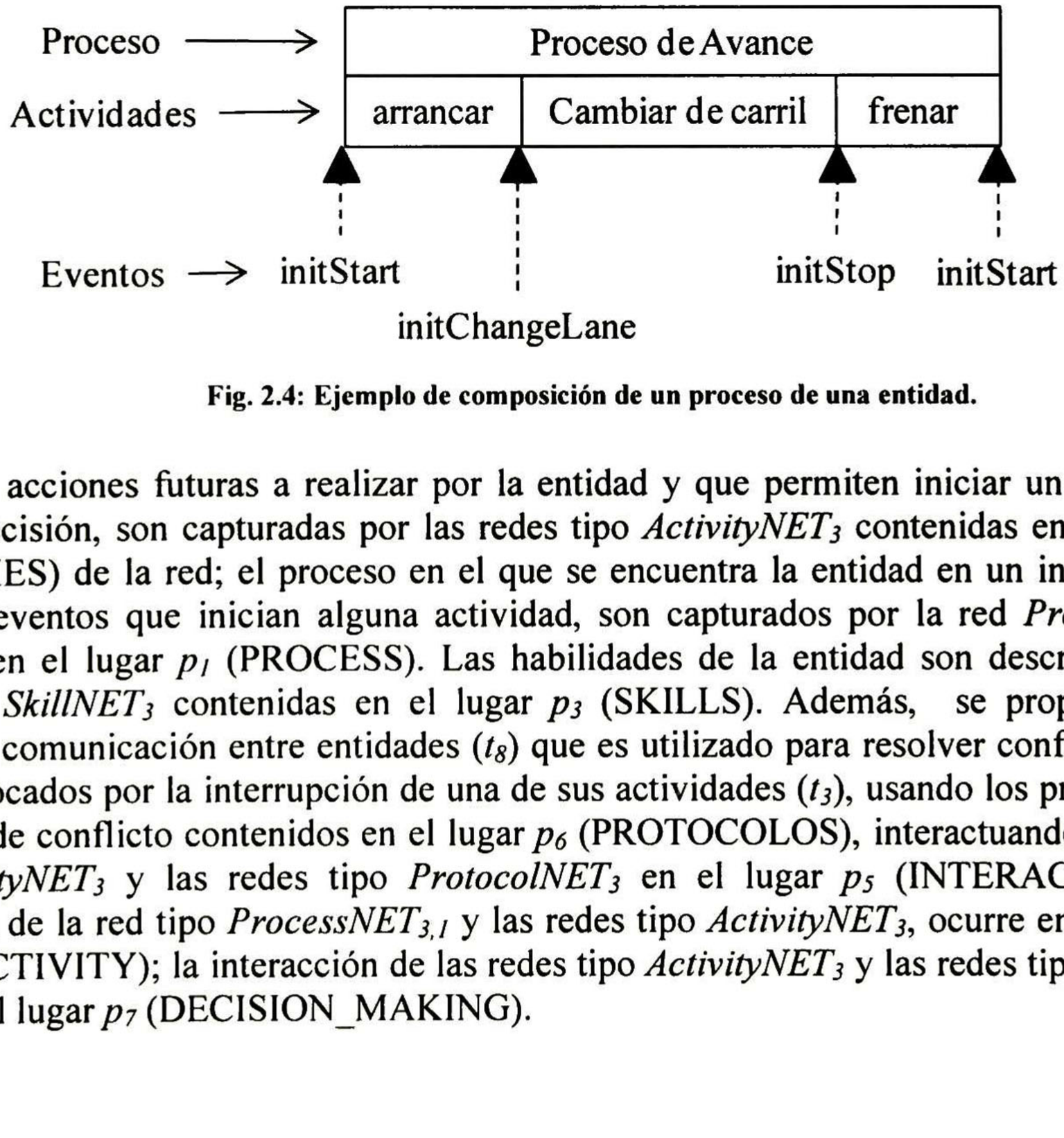


Fig. 2.4: Ejemplo de composición de un proceso de una entidad.

Las acciones futuras a realizar por la entidad y que permiten iniciar un proceso de toma de decisión, son capturadas por las redes tipo *ActivityNET*$_3$ contenidas en el lugar p_2 (ACTIVITIES) de la red; el proceso en el que se encuentra la entidad en un instante y los diferentes eventos que inician alguna actividad, son capturados por la red *ProcessNET*$_{3,1}$ contenida en el lugar p_1 (PROCESS). Las habilidades de la entidad son descritas por las redes tipo *SkillNET*$_3$ contenidas en el lugar p_3 (SKILLS). Además, se proporciona un módulo de comunicación entre entidades (t_8) que es utilizado para resolver conflictos entre ellas, provocados por la interrupción de una de sus actividades (t_3), usando los protocolos de colisión y de conflicto contenidos en el lugar p_6 (PROTOCOLOS), interactuando con la red tipo *ActivityNET*$_3$ y las redes tipo *ProtocolNET*$_3$ en el lugar p_5 (INTERACTING). La interacción de la red tipo *ProcessNET*$_{3,1}$ y las redes tipo *ActivityNET*$_3$, ocurre en el lugar p_4 (EXEC_ACTIVITY); la interacción de las redes tipo *ActivityNET*$_3$ y las redes tipo *SkillNET*$_3$ ocurre en el lugar p_7 (DECISION_MAKING).

Cada vez que la entidad requiere de alguna habilidad para desplazarse en la estructura de red vial, representada por la red de nivel 1, toma del lugar p_3 una habilidad, disparando la transición t_6 (*useSkill*), iniciando entonces un proceso de toma de decisión (PTD). Cuando la entidad ha finalizado su PTD, cada una de las acciones futuras a realizar, generadas durante este proceso, son asociadas al disparo de alguna de las transiciones de la red $ProcessNET_{3,1}$, lo que causará la ejecución de alguna nueva actividad, al disparar de nueva cuenta la transición t_3 (*initActivity*). En la Tabla 2.2 se presenta la descripción de la red tipo $EntityNET_{2,1}$.

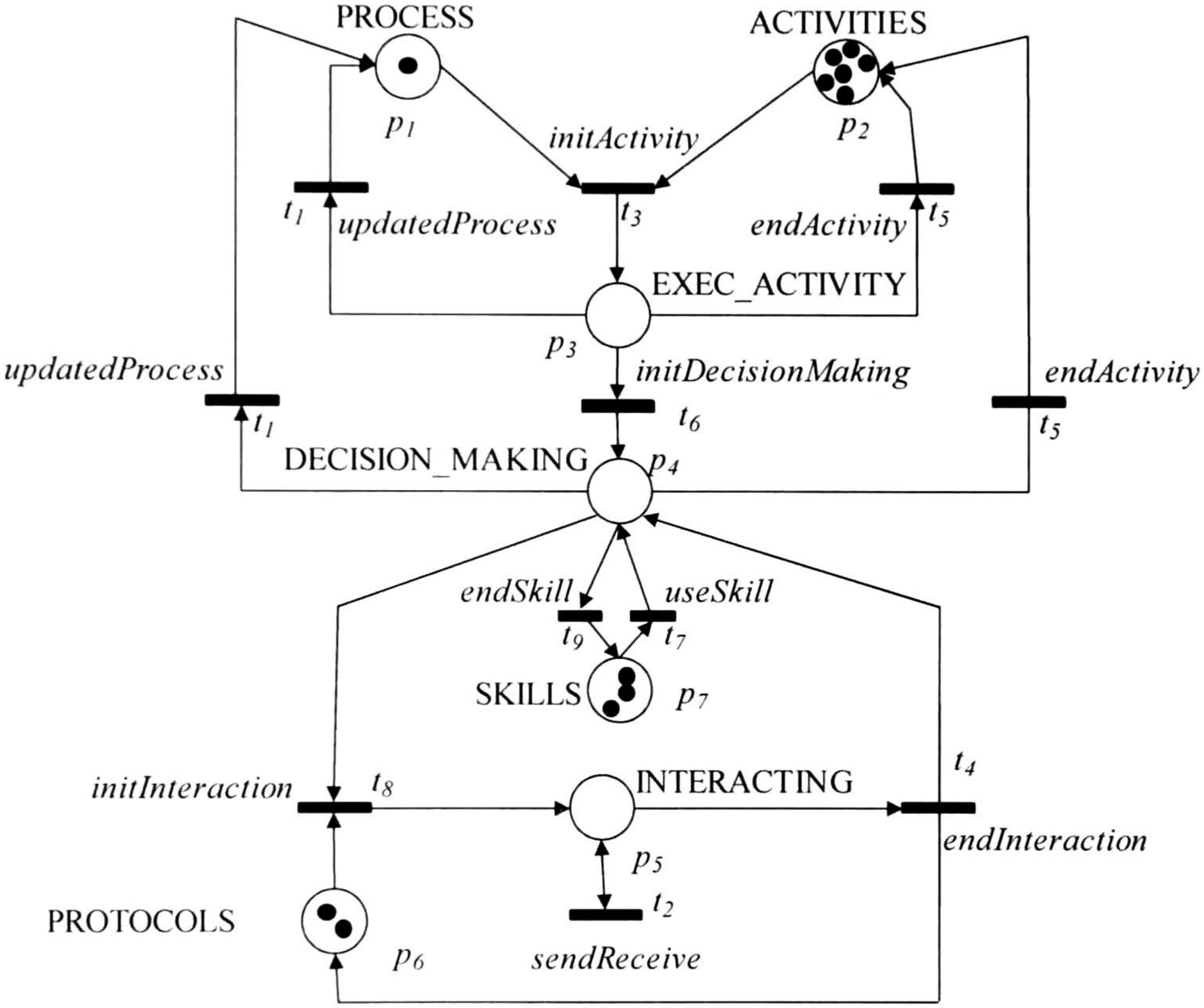


Fig. 2.5: Representación en el modelo de la estructura del entidad móvil $EntityNET_{2,1}$

$TypeEntityNET_{2,1}$

$TOKEN_{2,1}$ = {$ProcessNET_{3,1}$, $StartActivityNet_{3,2}$, $StopActivityNet_{3,3}$, $SenseAdvancingActivityNet_{3,4}$, $SenseStoppedActivityNet_{3,5}$, $ChangeLaneActivityNet_{3,6}$, $LeaveSegmentActivityNet_{3,7}$, $ArrivalLaneActivityNet_{3,8}$ $ArrivalSegmentActivityNet_{3,9}$, $SkillFreeFlowNet_{3,10}$ $SkillCarFollowingNet_{3,11}$ $SkillChangeLaneNet_{3,12}$ $ProtocolCollisionNet_{3,13}$, $ProtocolCollisionNet_{3,14}$ }

$LABEL_{2,1}$ = { *initArrivalSegment, initSenseAdvancing, initLeaveSegment, initChangeLane, initArrivalLane, initStart, initSenseStopped, initStop,initInConflict, finishActivity, sendReceive, initInteraction, endActivity, useSkillFreeFlow, useSkillCarFollowing, useSkillChangeLane, unusedSkill, updatedProcess*};

$VAR_{2,1}$ = {**x** : $TypeActivityNet_3$, **y**: $TypeProcessNET_{3,1}$, **z**: $TypeSkillNET_3$}

λ
$\lambda(t_1)=\{(updatedProcess^{\downarrow})\}$, $\lambda(t_2)=\{(sendReceive^{\equiv})\}$, $\lambda(t_3)=\{(initArrivalSegment^{\downarrow})$, $(initSenseAdvancing^{\downarrow})$, $(initLeaveSegment^{\downarrow})$, $(initChangeLane^{\downarrow})$, $(initArrivalLane^{\downarrow})$, $(initStart^{\downarrow})$, $(initSenseStopped^{\downarrow})$, $(initStop^{\downarrow})\}$, $\lambda(t_4)=\{(endInteraction^{\equiv})\}$, $\lambda(t_5)=\{(endActivity^{\downarrow})\}$, $\lambda(t_6)=\{(useSkillFreeFlow^{\downarrow})$, $(useSkillCarFollowing^{\downarrow})$, $(useSkillChangeLane^{\downarrow})\}$, $\lambda(t_7)=\{(endSkill^{\downarrow})\}$, $\lambda(t_8)=\{(initInteraction^{\equiv})\}$
τ
$\tau(p_1) = \{ ProcessNET_{3,1}\}$, $\tau(p_2) = \{ StartActivityNet_{3,2}$, $StopActivityNet_{3,3}$, $SenseAdvancingActivityNet_{3,4}$, $SenseStoppedActivityNet_{3,5}$, $ChangeLaneActivityNet_{3,6}$, $LeaveSegmentActivityNet_{3,7}$, $ArrivalLaneActivityNet_{3,8}$, $ArrivalSegmentActivityNet_{3,9}\}$, $\tau(p_3) = \{ SkillFreeFlowNet_{3,10}$ $SkillCarFollowingNet_{3,11}$ $SkillChangeLaneNet_{3,12}\}$, $\tau(p_4) = \{ ProcessNET_{3,1}$, $StartActivityNet_{3,2}$, $StopActivityNet_{3,3}$, $SenseAdvancingActivityNet_{3,4}$, $SenseStoppedActivityNet_{3,5}$, $ChangeLaneActivityNet_{3,6}$, $LeaveSegmentActivityNet_{3,7}$, $ArrivalLaneActivityNet_{3,8}$ $ArrivalSegmentActivityNet_{3,9}\}$,$\tau(p_5) = \{ StartActivityNet_{3,2}$, $StopActivityNet_{3,3}$, $SenseAdvancingActivityNet_{3,4}$, $SenseStoppedActivityNet_{3,5}$, $ChangeLaneActivityNet_{3,6}$, $LeaveSegmentActivityNet_{3,7}$, $ArrivalLaneActivityNet_{3,8}$, $ArrivalSegmentActivityNet_{3,9}$, $ProtocolConflictNET_{3,13}$, $ProtocolCollisionNET_{3,14}\}$, $\tau(p_6) = \{ProtocolConflictNET_{3,13}$, $ProtocolCollisionNET_{3,14}\}$, $\tau(p_7) = \{ SkillFreeFlowNet_{3,10}$, $SkillCarFollowingNet_{3,11}$, $SkillChangeLaneNet_{3,12}$, $ProtocolConflictNET_{3,13}$, $ProtocolCollisionNET_{3,14}\}$
π
$\pi((p_4, t_1), updatedProcess^{\downarrow}) = \pi((t_1, p_1), updatedProcess^{\downarrow}) = \pi((p_1, t_3), initStart^{\downarrow}) = \pi((p_1, t_3), initChangeLane^{\downarrow}) = \pi((p_1, t_3), initStop^{\downarrow}) = \pi((p_1, t_3), initSenseStopped^{\downarrow}) = \pi((p_1, t_3), initArrivalSegment^{\downarrow}) = \pi((p_1, t_3), initSenseAdvancing^{\downarrow}) = \pi((p_1, t_3), initLeaveSegment^{\downarrow}) = \pi((p_1, t_3), initArrivalLane^{\downarrow}) = \{\mathbf{y}\}$, $\pi((t_3, p_4), initStart^{\downarrow}) = \pi((t_3, p_4), initStop^{\downarrow}) = \pi((t_3, p_4), initSenseStopped^{\downarrow}) = \pi((t_3, p_4), initArrivalSegment^{\downarrow}) = \pi((t_3, p_4), iniSenseAdvancing^{\downarrow}) = \pi((t_3, p_4), initLeaveSegment^{\downarrow}) = \pi((t_3, p_4), initArrivalLane^{\downarrow}) = \{\mathbf{x+y}\}$, $\pi((p_4, t_5), initStart^{\downarrow}) = \pi((t_5, p_2), initStart^{\downarrow}) = \pi((p_2, t_3), initStart^{\downarrow}) = \pi((p_4, t_5), initChangeLane^{\downarrow}) = \pi((t_5, p_2), initChangeLane^{\downarrow}) = \pi((p_2, t_3), initChangeLane^{\downarrow}) = \pi((p_4, t_5), initSenseStopped^{\downarrow}) = \pi((t_5, p_2), initSenseStopped^{\downarrow}) = \pi((p_2, t_3), initSenseStopped^{\downarrow}) = \pi((p_4, t_5), initStop^{\downarrow}) = \pi((t_5, p_2), initStop^{\downarrow}) = \pi((p_2, t_3), initStop^{\downarrow}) = \pi((p_4, t_5), initArrivalSegment^{\downarrow}) = \pi((t_5, p_2), initArrivalSegment^{\downarrow}) = \pi((p_2, t_3), initArrivalSegment^{\downarrow}) = \pi((p_4, t_5), initSenseAdvancing^{\downarrow}) = \pi((t_5, p_2), initSenseAdvancing^{\downarrow}) = \pi((p_2, t_3), initSenseAdvancing^{\downarrow}) = \pi((p_4, t_5), initLeaveSegment^{\downarrow}) = \pi((t_5, p_2), initLeaveSegment^{\downarrow}) = \pi((p_2, t_3), initLeaveSegment^{\downarrow}) = \pi((p_4, t_5), initArrivalLane^{\downarrow}) = \pi((t_5, p_2), initArrivallane^{\downarrow}) = \pi((p_2, t_3), initArrivalLane^{\downarrow}) = \{\mathbf{x}\}$, $\pi((p_4, t_6), useSkill^{\downarrow}) = \{\mathbf{x}\}$, $\pi((t_6, p_7), useSkill^{\downarrow}) = \{\mathbf{x+z}\}$, $\pi((p_3, t_6), useSkill^{\downarrow}) = \{\mathbf{z}\}$, $\pi((p_7, t_7), endSkill^{\downarrow}) = \{\mathbf{x+z}\}$, $\pi((t_7, p_3), endSkill^{\downarrow}) = \{\mathbf{z}\}$, $\pi((t_7, p_4), endSkill^{\downarrow}) = \{\mathbf{x}\}$,

Tabla 2.2: Funciones λ,τ and π de la red *EntityNET*$_{2,1}$

Comportamiento específico de las entidades

El comportamiento individual de una entidad en el modelo n-LNS, es descrito por las redes de tipo *ProcessNET*$_3$, *ActivitiyNET*$_3$ y *SkillNET*$_3$, que sincronizadas por la red *EntitiyNET*$_2$, permiten capturar diferentes tipos de comportamientos en el modelo.

Red *ProcessNET*$_3$

Las entidades exhiben comportamientos únicos originados en parte por sus características propias y el grado de conocimiento del dominio en el que están situadas. La siguiente definición captura esta propiedad:

Definición 2.10. La descripción de una entidad *i* es un 4-tupla definida por $\boldsymbol{entity_i} = (ATTR_i, PARAM_i, knowledgeBase_i, plan_i)$ donde:

- $ATTR_i$, es un conjunto de atributos de la entidad que describen la situación de la entidad *i*. $ATTR_i$ = {*velocidad, hambre, humor, posición*}.
- $PARAM_i$ es un conjunto de parámetros de la entidad. $PARAM_i$= {*tiempoPercepciónReaccion, capacidad, tamaño, color, plan, factorAgresividad, velocidadMaxima, riesgoRebasar*}.
- $knowledgeBase_i$ es la base de conocimiento de la entidad, formada de reglas con el formato *if-then*. La metodología de captura de estas reglas será descrita más adelante.
- $plan_i$ es el plan de la entidad descrito por una secuencia de segmentos s_i.

La red $ProcessNET_{3,1}$ describe los procesos en los que se puede encontrar una entidad, así como los eventos que pueden ser ejecutados en cada uno de ellos y los eventos que permiten cambie entre procesos (ver Fig. 2.6). La marca del lugar p_1 de la red $ProcessNET_{3,1}$ es asociada a la descripción de la entidad $\boldsymbol{entity_i}$. En la Tabla 2.3 se describen los detalles de la red.

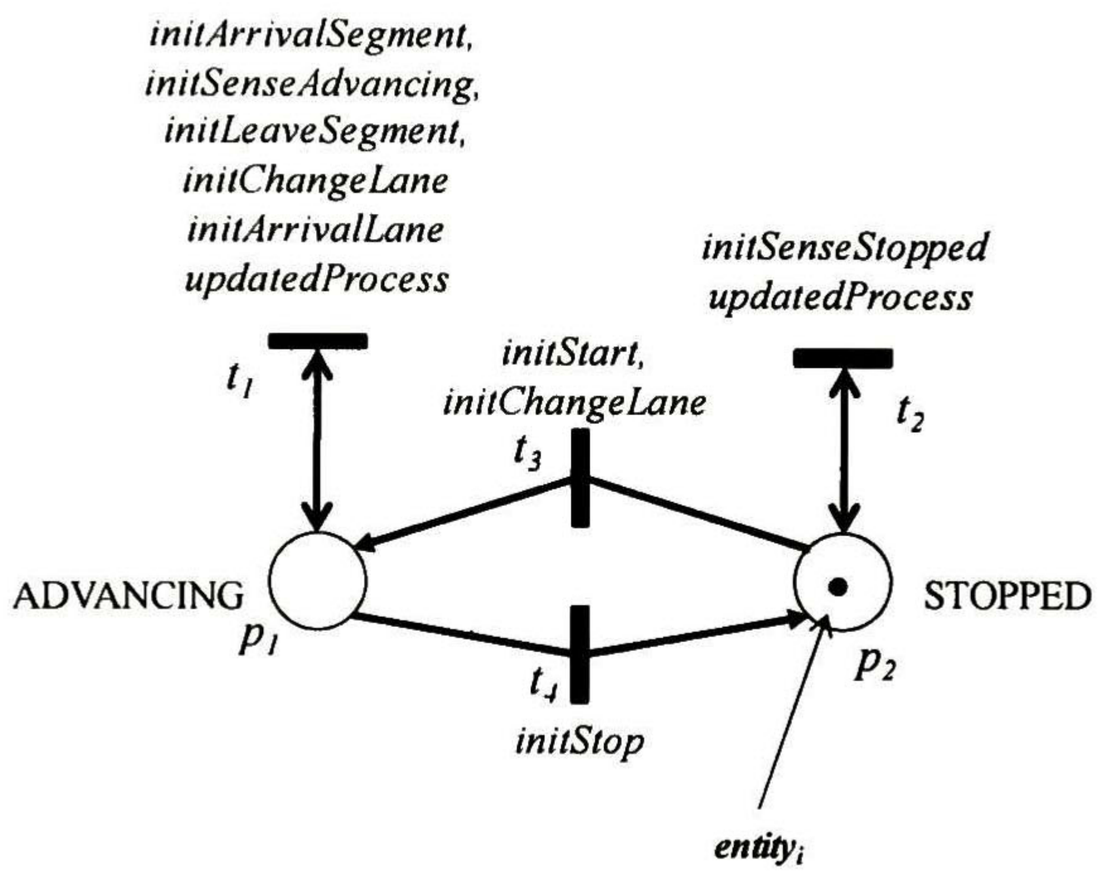


Fig. 2.6: Representación en el modelo de los procesos y eventos del vehículo $ProcessNET_{3,1}$

$\boldsymbol{TypeProcessNET_{3,1}}$
$TOKEN_{3,1}$ = {*entity*$_i$}; $LABEL_{3,1}$ = {*initArrivalSegment, initSenseAdvancing, initLeaveSegment, initChangeLane, initArrivalLane, initStart, initSenseStopped, initStop, updatedProcess* }
λ
$\lambda(t_1)$={(*arrivalSegment*$^{\uparrow}$), (*initSenseAdvancing*$^{\uparrow}$), (*initLeaveSegment*$^{\uparrow}$), (*initChangeLane*$^{\uparrow}$), (*initArrivalLane*$^{\uparrow}$)}, $\lambda(t_2)$={(*initSenseStopped*,$^{\uparrow}$)}, $\lambda(t_3)$={(*initStart*$^{\uparrow}$), (*initChangeLane*$^{\uparrow}$)}, $\lambda(t_4)$={(*initStop*$^{\uparrow}$)},
τ
$\tau(p_1) = \tau(p_2)$ ={*entity*$_i$}
π

$\pi((p_1, t_1), initArrivalSegment^{\uparrow})=\pi((t_1, p_1), initArrivalSegment^{\uparrow})= \pi((p_1, t_1), initSenseAdvancing^{\uparrow})=\pi((t_1, p_1), initLeaveSegment^{\uparrow})= \pi((p_1, t_1), initLeaveSegment^{\uparrow})=\pi((t_1, p_1), initSenseAdvancing^{\uparrow}) = \pi((t_1, p_1), initChangeLane^{\uparrow})= \pi((p_1, t_1), initChangeLane^{\uparrow})=\pi((t_1, p_1), initArrivalLane^{\uparrow})= \pi((p_1, t_1), initArrivalLane^{\uparrow})=\pi((p_2, t_2), initSenseStopped^{\uparrow})=\pi((t_2, p_2), initSenseStopped^{\uparrow})=\pi((p_2, t_3), initStart^{\uparrow})=\pi((t_3, p_1), initStart^{\uparrow})= \pi((p_2, t_3), initChangeLane^{\uparrow})=\pi((t_3, p_1), initChangeLane^{\uparrow})= \pi((p_1, t_4), initStop^{\uparrow})=\pi((t_4, p_2), initStop^{\uparrow}) = entity_i$

Tabla 2.3: Funciones λ,τ and π de la red *ProcessNET*$_{3,1}$

Red *ActivityNET*$_3$

Un entidad *i* tiene definidas n_i actividades que puede realizar en el ambiente $\{a_{i1}, a_{i2}, \ldots, a_{in_i}\}$ **[102]**. Manteniendo una gráfica de las actividades de la entidad ejecutadas en el ambiente con respecto a alguna variable de la propia entidad, podemos definir el comportamiento de la entidad en el ambiente. La percepción de la secuencia de ejecución de estas actividades es llamada fuente de datos de comportamiento **[103]**. Cada una de las n_i actividades de la entidad es iniciada por algún evento de de la red *ProcessNET*$_{3,1}$.

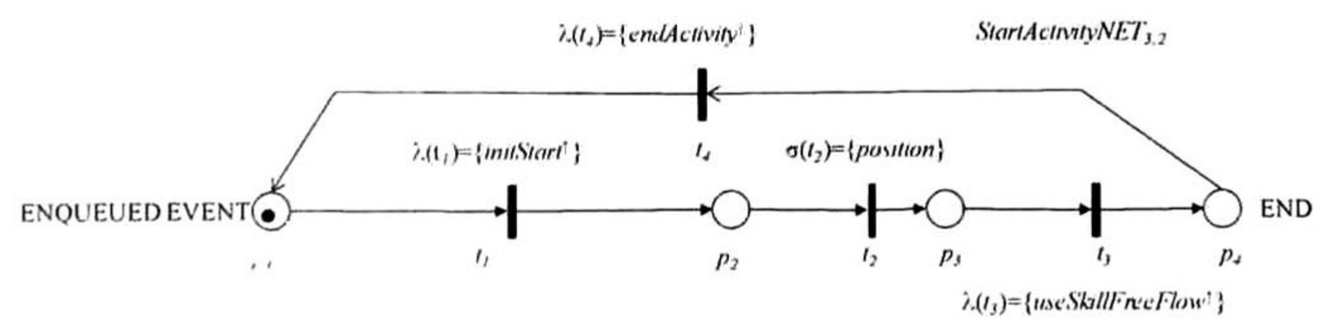
Estas actividades son capturadas en una red de nivel 3 del formalismo n-LNS, para el caso del vehículo se han definido las siguientes actividades: *StartActivityNet*$_{3,2}$, *StopActivityNet*$_{3,3}$, *SenseAdvancingActivityNet*$_{3,4}$, *SenseStoppedActivityNet*$_{3,5}$, *ChangeLaneActivityNet*$_{3,6}$, *LeaveSegmentActivityNet*$_{3,7}$, *ArrivalLaneActivityNet*$_{3,8}$, *ArrivalSegmentActivityNet*$_{3,9}$.

Cada una de estas redes modifica alguno de los atributos de la entidad, esto es capturado en el modelo con la siguiente función:

Definición 2.11. σ:T→ATTR$_i$ es una función de asignación de atributos de la entidad a algunas transiciones de la red tipo *ActivityNET*$_3$, lo que permite saber que atributo es modificado cuando dicha transición es disparada, donde:

- T es el conjunto de transiciones de la red *ActivityNET*$_3$.
- ATTR$_i$ es el conjunto de atributos de la entidad *i*.

A manera de ejemplo, en la Fig. 2.6 se describen las redes *StartActivityNET*$_{3,2}$ y *StopActivityNET*$_{3,3}$. En la Tabla 2.4: se muestra las funciones de sincronización para la red *StopActivityNET*$_{3,3}$.



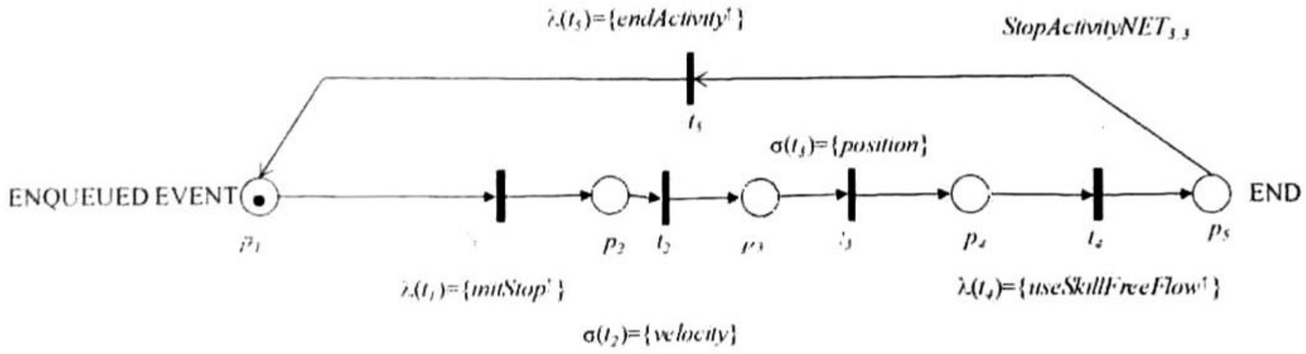


Fig. 2.7: Redes de actividades de la entidad *StartActivityNET*$_{3,2}$ y *StopActivityNET*$_{3,3}$

TypeStopActivityNET$_{3,3}$
$TOKEN_{3,3}$ = {*} $LABEL_{3,3}$ = {*initStop, useSkillFreeFlow, endActivity* }
λ
$\lambda(t_1)=\{(initStop^{\uparrow})\}$, $\lambda(t_4)=\{(useSkillFreeFlow^{\uparrow})\}$, $\lambda(t_5)=\{(endActivity^{\uparrow})\}$
τ
$\tau(p_1) = \tau(p_2) = \tau(p_3) = \tau(p_4) = \tau(p_5) = \{*\}$
π
$\pi((p_1, t_1), initStop^{\uparrow})=\pi((t_1, p_2), initStop^{\uparrow}) = \pi((p_4, t_4), useSkillFreeFlow^{\uparrow})=\pi((t_4, p_5), useSkillFreeFlow^{\uparrow})= \pi((p_5, t_5), endActivity^{\uparrow})=\pi((t_5, p_1), endActivity^{\uparrow}) = *$
σ
$\sigma(t_2)=\{velocity\}$, $\sigma(t_3)=\{position\}$

Tabla 2.4: Funciones λ, π y σ de la red *TypeStopActivityNET$_{3,3}$*

Red *SkillNET$_3$*

Una entidad posee ciertas habilidades que le permiten desplazarse en su ambiente. Algunas de estas habilidades requieren del uso de un mecanismo de toma de decisión o de procedimientos que la permitan establecer el valor de ciertas variables necesarias durante el PTD. Para el caso del vehículo podemos describir las siguientes tres habilidades: avance-libre (*freeFlow*), seguimiento-vehículo (*carFollowing*) y cambio de carril (*changeLane*). Estas habilidades son descritas respectivamente por las redes *SkillFreeFlowNET$_{3,10}$*, *SkillCarFollowingNET$_{3,11}$* y *SkillChangeLaneNET$_{3,12}$*.

Algunos de los eventos de las redes *SkillNET$_3$* permiten que la entidad inicie un PTD. Se ha agregado la función σ para describir aquellas transiciones que permiten se inicie un PTD:

Definición 2.12. $\varphi: T \rightarrow 2^{LABELi,k}$ es una función de asignación de inicio de toma de decisión de la entidad a algunas transiciones de las redes *SkillNET$_3$*.

SkillFreeFlowNET$_{3,10}$
$TOKEN_{3,10}$ = {*}; $LABEL_{3,1}$ = {*useSkillFreeFlow, searchObstacle, initSenseStopped, newStart, newSenseAdvancing, updatedProcess, endSkill, useSkillCarFollowing* }
λ
$\lambda(t_1)=\{(useSkillFreeFlow^{\uparrow})$, $\lambda(t_2)=\{(searchObstacle^{\uparrow})\}$, $\lambda(t_5)=\{(initSenseStopped^{=})\}$, $\lambda(t_6)=\{(initSenseAdvancing^{=})\}$, $\lambda(t_7)=\{(useSkillCarFollowing^{\uparrow})\}$, $\lambda(t_8)=\{(newStart^{\uparrow})\}$, $\lambda(t_9)=\{(newSenseAdvancing^{\uparrow})\}$, $\lambda(t_{10})=\{(updatedProcess^{\uparrow})\}$, $\lambda(t_{11})=\{(endSkill^{\uparrow})\}$
τ
$\tau(p_1) = \tau(p_2) = \tau(p_3) = \tau(p_4) = \tau(p_5) = \tau(p_6) = \tau(p_7) = \tau(p_8) = \tau(p_9) = \{*\}$
π
$\pi((p_1, t_1), useSkillFreeFlow^{\uparrow})=\pi((t_1, p_2), useSkillFreeFlow^{\uparrow})= \pi((p_4, t_5), initSenseStopped^{=})=\pi((t_5, p_6), initSenseStopped^{=})= \pi((p_6, t_8), newStart^{\uparrow})=\pi((t_8, p_7), newStart^{\uparrow})= \pi((p_7, t_9), \text{newSenseAdvancing}^{\uparrow})=\pi((t_9, p_8), newSenseAdvancing^{\uparrow})= \pi((p_8, t_{10}), updatedProcess^{\uparrow})=\pi((t_{10}, p_9), updatedProcess^{\uparrow})= \pi((p_9, t_{11}), endSkill^{\uparrow})=\pi((t_{11}, p_1), \text{endSkill}^{\uparrow}) = *$
φ

$\varphi(t_2)=\{searchObstacle\}$

Tabla 2.5: Funciones λ,τ and π de la red *SkillFreeFlowNET*$_{3,10}$

Red *SkillFreeFlowNET*$_{3,10}$

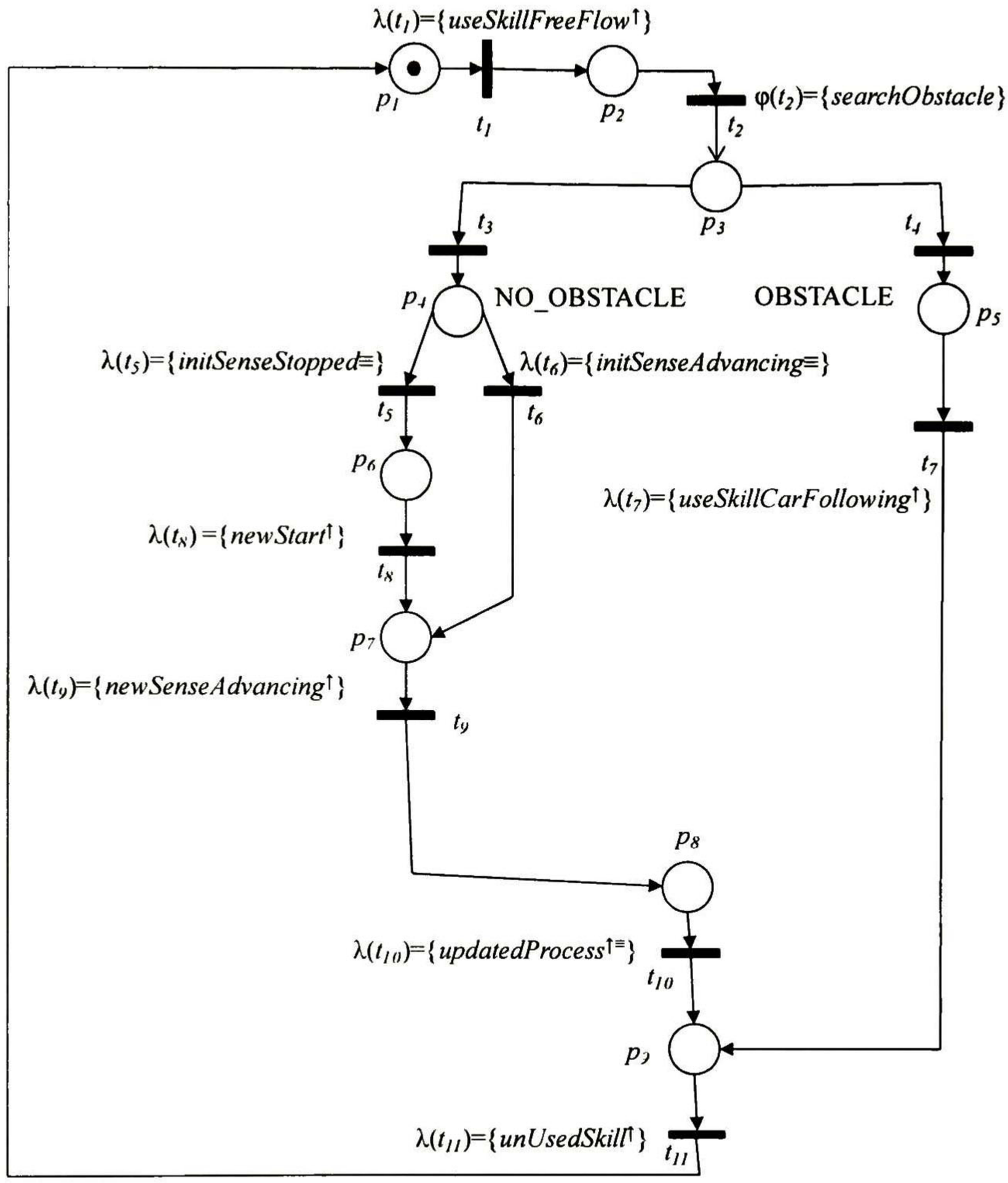


Fig. 2.8: *SkillFreeFlowNET*$_{3,10}$

Red $SkillCarFollowingNET_{3,1}$

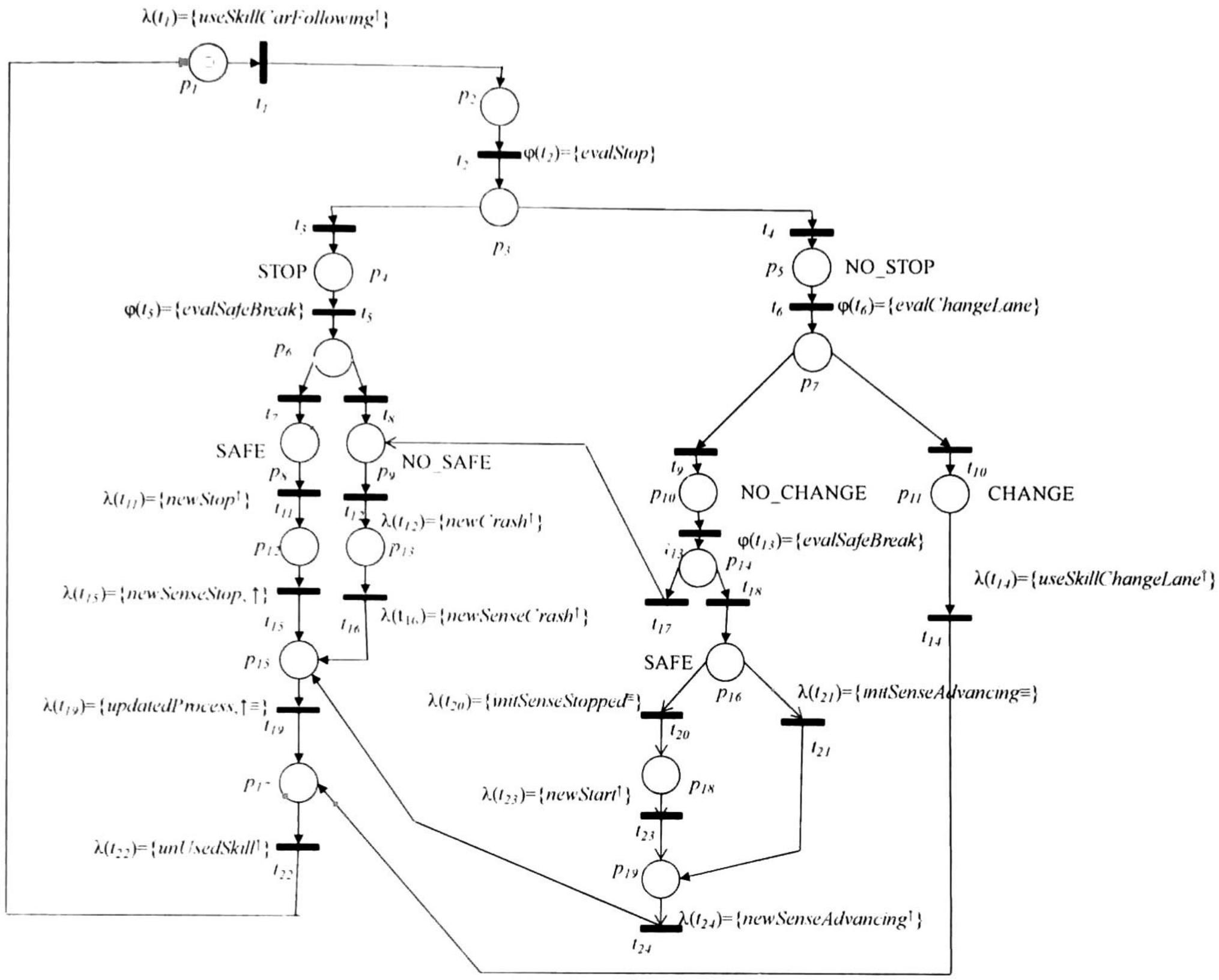


Fig. 2.9: $SkillCarFollowingNET_{3,11}$

$SkillCarFollowingNET_{3,11}$

$TOKEN_{3,11}$ = {*}; $LABEL_{3,11}$ = {*useSkillCarFollowing, evalStop, evalSafeBreak, newStop, newSenseStop, updatedProcess, endSkill, newCrash, newSenseCrash, evalChangeLane, evalSafeBreak, initSenseStopped, newStart, initSenseAdvancing, newSenseAdvancing, useSkillChangeLane* }

λ

$\lambda(t_1)=\{(useSkillCarFollowing^{\uparrow}),\ \lambda(t_2)=\{((evalStop^{\uparrow})\},\ \lambda(t_5)=\{(evalSafeBreak^{\uparrow})\},\ \lambda(t_{11})=\{(newStop^{\uparrow}),$
$\lambda(t_{12})=\{((newCrash^{\uparrow}),\ \lambda(t_{13})=\{((evalSafeBreak^{\uparrow}),\ \lambda(t_{14})=\{((useSkillChangeLane^{\uparrow}),\ \lambda(t_{15})=\{((newSenseStop^{\uparrow}),$
$\lambda(t_{16})=\{((newSenseCrash^{\uparrow}),\ \lambda(t_{19})=\{((updatedProcess^{\uparrow}),\ \lambda(t_{20})=\{((initSenseStopped^{\equiv}),$
$\lambda(t_{21})=\{((initSenseAdvancing^{\equiv}),\ \lambda(t_{22})=\{((endSkill^{\uparrow}),\ \lambda(t_{23})=\{((newStart^{\uparrow}),\ \lambda(t_{24})=\{((newSenseAdvancing^{\uparrow}),$

τ

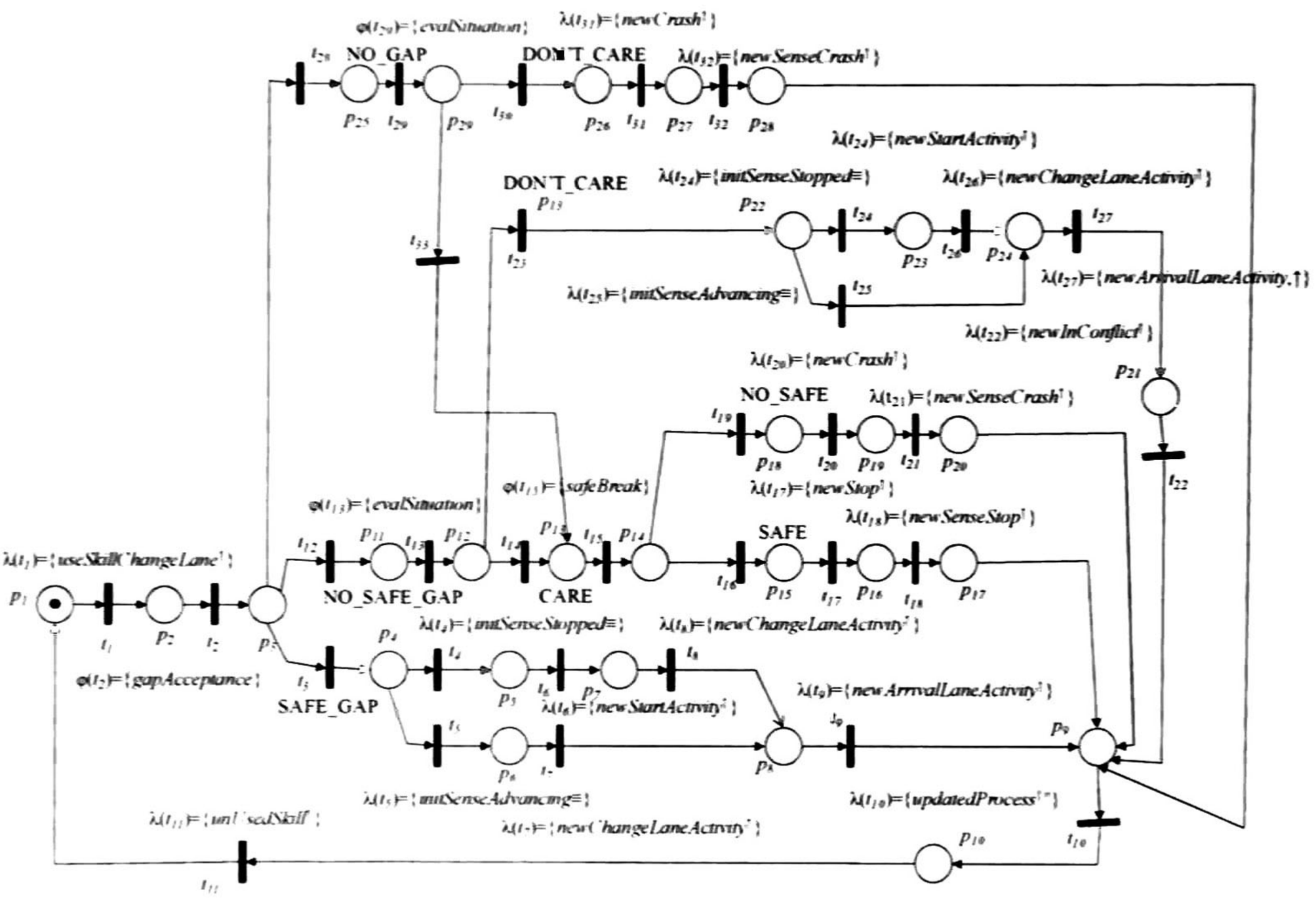
$\tau(p_1) = \tau(p_2) = \tau(p_3) = \tau(p_4) = \tau(p_5) = \tau(p_6) = \tau(p_7) = \tau(p_8) = \tau(p_9) = \tau(p_{10}) = \tau(p_{11}) = \tau(p_{12}) = \tau(p_{13}) = \tau(p_{14}) = \tau(p_{15}) = \tau(p_{16}) = \tau(p_{17}) = \tau(p_{18}) = \tau(p_{19}) = \{*\}$

π

$\pi((p_1, t_1), useSkillCarFollowing^{\uparrow}) = \pi((t_1, p_1), useSkillCarFollowing^{\uparrow}) = \pi((p_8, t_{11}), newStop^{\uparrow}) = \pi((t_{11}, p_{12}), newStop^{\uparrow}) = \pi((p_{12}, t_{15}), newSenseStop^{\uparrow}) = \pi((t_{15}, p_{15}), newSenseStop^{\uparrow}) = \pi((p_1, t_1), arrival^{\uparrow}) = \pi((t_1, p_1), initChangeLane^{\uparrow}) = \pi((p_1, t_1), initChangeLane^{\uparrow}) = \pi((t_1, p_1), arrivalChangeSegment^{\uparrow}) = \pi((p_1, t_1), arrivalChangeSegment^{\uparrow}) = \pi((p_2, t_2), initSenseStopped^{\uparrow}) = \pi((t_2, p_2), initSenseStopped^{\uparrow}) = \pi((p_2, t_3), start^{\uparrow}) = \pi((t_3, p_1), start^{\uparrow}) = \pi((p_2, t_3), initChangeLane^{\uparrow}) = \pi((t_3, p_1), initChangeLane^{\uparrow}) = \pi((p_1, t_4), initStop^{\uparrow}) = \pi((t_4, p_2), initStop^{\uparrow}) = *$
φ
$\varphi(t_2)=\{(evalStop)\}, \varphi(t_5)=\{(evalSafeBreak)\}, \varphi(t_6)=\{(evalChangeLane)\}, \varphi(t_{13})=\{(evalSafeBreak)\}.$

Tabla 2.6: Funciones λ,τ and π de la red *SkillCarFollowingNET*$_{3,11}$

Red *SkillChangeLaneNET*$_{3,12}$



2.10: Red *SkillChangeLaneNET*$_{3,12}$.

***SkillChangeLaneNET*$_{3,12}$**
$TOKEN_{3,12} = \{*\}$: $LABEL_{3,12} = \{useSkillChangeLane, arrivalSegment, initSenseAdvancing, initLeaveSegment, arrival, initChangeLane, arrivalChangeSegment, start, initSenseStopped, initStop\}$
λ
$\lambda(t_3)=\{(start, \uparrow), (initChangeLane, \uparrow)\}, \lambda(t_4)=\{(initStop, \uparrow)\}, \lambda(t_1)=\{(arrivalSegment, \uparrow), (initSenseAdvancing, \uparrow), (initLeaveSegment, \uparrow), (arrival, \uparrow), (initChangeLane, \uparrow), (arrivalChangeSegment, \uparrow)\}, \lambda(t_2)=\{(initSenseStopped, \uparrow)\}$
τ
$\tau(p_1) = \tau(p_2) = \tau(p_3) = \tau(p_4) = \tau(p_5) = \tau(p_6) = \tau(p_7) = \tau(p_8) = \tau(p_9) = \tau(p_{10}) = \tau(p_{11}) = \tau(p_{12}) = \tau(p_{13}) =$

$\tau(p_{14}) = \tau(p_{15}) = \tau(p_{16}) = \tau(p_{17}) = \tau(p_{18}) = \tau(p_{19}) = \tau(p_{20}) = \tau(p_{21}) = \tau(p_{22}) = \tau(p_{23}) = \tau(p_{24}) = \tau(p_{25}) = \tau(p_{26}) = \tau(p_{27}) = \tau(p_{28}) = \{*\}$
π
$\pi((p_1, t_1), useSkillChangeLane^{\uparrow}) = \pi((t_1, p_1), useSkillChangeLane^{\uparrow}) = \pi((p_4, t_5), initSenseStopped^{=}) = \pi((t_5, p_6), initSenseStopped^{=}) = \pi((p_6, t_8), newStart^{\uparrow}) = \pi((t_8, p_7), newStart^{\uparrow}) = \pi((p_7, t_9), \text{newSenseAdvancing}^{\uparrow}) = \pi((t_9, p_8), newSenseAdvancing^{\uparrow}) = \pi((p_8, t_{10}), updatedProcess^{\uparrow}) = \pi((t_{10}, p_9), updatedProcess^{\uparrow}) = \pi((p_9, t_{11}), endSkill^{\uparrow}) = \pi((t_{11}, p_1), \text{endSkill}^{\uparrow}) = *$
φ
$\varphi(t_2) = \{(gapAcceptance)\}, \varphi(t_{13}) = \{(evalSituation)\}, \varphi(t_{15}) = \{(safeBreak)\}, \varphi(t_{29}) = \{(evalSituation)\}$

Tabla 2.7: Funciones λ,τ and π de la red *SkillChangeLaneNET*$_{3,12}$

2.2.4. Ejemplo de Sincronización del modelo n-LNS del STU

Con el fin de mostrar la sincronización de las redes del modelo n-LNS supondremos el marcado de la red $EntityNET_{2,1}$ mostrado en la Fig. 2.5. La primer actividad ejecutada es la descrita por la red $StartActivityNET_{3,2}$, esto es la transición t_3 de la red $ProcessNET_{3,1}$ es sincronizada con la transición t_1 de la red $StartActivityNET_{3,2}$ a través de la transición t_3 de la red $EntityNET_{2,1}$ marcando entonces el lugar p_4 de la red $EntityNET_{2,1}$ con las redes $ProcessNET_{3,1}$ y $StartActivityNET_{3,2}$.

La red $StartActivityNET_{3,2}$ modifica la posición y velocidad de la entidad i ($entity_i$), se sincroniza con la transición t_6 (*useSkill*), requiriendo de esta manera el uso de la red $SkillFreeFlowNET_{3,10}$. Esta sincronización marca el lugar p_7 (DECISION_MAKING) de la red $EntityNET_{2,1}$ con la red $StartActivityNET_{3,2}$ y la red $SkillFreeFlowNET_{3,10}$. Si el uso de esta habilidad permite definir las siguientes actividades a realizar por la entidad, se regresa la habilidad al lugar de habilidades y la actividad al lugar p_4 de la red $EntityNET_{2,1}$ al sincronizar estas redes con la transición t_5 (*endSkill*) de la red $EntityNET_{2,1}$. De requerir otra habilidad para definir las actividades siguientes, se vuelve a marcar el lugar de toma de decisión con la habilidad solicitada.

La red $SkillFreeFlowNET_{3,10}$ permite ubicar los obstáculos del segmento en el que se encuentra contenida la entidad. En el caso que no existan obstáculos, las siguientes transiciones de la red $ProcessNET_{3,1}$ que se dispararán son las etiquetadas con *initLeaveSegment* e *initArrival,* de lo contrario usa la red $SkillCarFollowingNET_{3,11}$ para establecer las siguientes actividades.

Suponiendo que no existan obstáculos, las siguiente red de actividad que se colocará junto con la red $ProcessNET_{3,1}$ en el lugar p_4 de la red $EntityNET_{2,1}$ será la red $LeaveSegmentActivityNET_{3,7}$ La posterior ejecución de la red $LeaveSegmentActivityNET_{3,7}$ sincronizará la red $EntityNET_{2,1}$ con una transición de la red de nivel 1 etiquetada con *initLeaveSegment* , que permitirá que la entidad se desplace a otro segmento.

Una vez decididas las siguientes actividades que realizará la entidad, la red $ProcessNET_{3,1}$ retorna al lugar p_1 (PROCESS), sincronizando las transiciones de la red $ProcessNET_{3,1}$, $StartActivityNET_{3,2}$ y $EntityNET_{2,1}$ etiquetadas con *updatedProcess*.

Mecanismo de toma de decisión

El mecanismo de toma de decisión (MTD) asociado a algunas transiciones de las actividades de la entidad por la función φ, permite que la entidad pueda realizar toma de decisiones que la lleven a lograr sus objetivos, usando el estado de sí mismo y su ambiente. La capacidad de razonamiento de la entidad puede ser modelada utilizando razonamientos basados en reglas, redes bayesianas, lógica difusa, etc. En este trabajo se utiliza el razonamiento basado en reglas. Las entidades descritas de esta manera tienen una arquitectura que consiste en tres componentes: la base de conocimiento (saber acerca del dominio en el que se encuentra), base de hechos (parámetros, atributos de la entidad y eventos de otras entidades en el ambiente) y las habilidades requeridas por la entidad en la toma de decisión.

La base de conocimientos

El nivel de conocimiento del dominio en el que se encuentra la entidad, que le permite a la entidad poder realizar una toma de decisión, es capturado en la base de conocimiento de la entidad $knowledgeBase_i$. Ésta es expresada como un conjunto de reglas *if-then*. Las reglas tienen el siguiente formato:

$$if\ (COND_1[AND\ COND_2 \ldots AND\ COND_n])$$
$$then(evt_1(time, position, s_i, entity_i)[$$
$$AND\ evt_2(time, position, s_i, entity_i) \ldots$$
$$AND\ evt_n(time, position, s_i, entity_i)]$$

Donde:

- $COND_n$ describe el estado interno de la entidad o del ambiente. $COND_n$ puede ser sustituida con las siguientes condiciones: (*SpatialSimulation* = *entityType*), (*eventObstacle* = evt_i), (*temporalRelation* = *before*), (*temporalRelation* = *after*), (*safeChange* = *false*), (*safeChange* = *true*).
- $evt_i(time, position, s_i, entity_i)$ describe el evento futuro que será ejecutado cuando la condición (condiciones) de la regla sea verdadera.

Habilidades

En los modelos dirigidos por la técnica de avance de simulación de TD, la entidad toma una decisión con respecto a lo que pasó un instante antes, y tiene acceso directo al estado actual del ambiente, como es el caso de los modelos vehículo-siguiente con la técnica de TD [**32**][**104**][7]. En el modelo de STU n-LNS la técnica de avance es de ED, entonces el tiempo sobre el que la entidad toma un decisión es variable, además, la entidad no tiene acceso de manera directa al estado futuro de las variables del sistema (o de otras entidades), así que tendrá que calcularlas. Los procesos necesarios para realizar estos cálculos son llamados habilidades y ayudan en la toma de decisión de la entidad. Estos procesos retornan

el valor de alguna de las condiciones $COND_i$ de las reglas de la base de conocimiento de la entidad.

Hechos

La base de datos o los hechos conocidos por la entidad, son los atributos y parámetros de la entidad, así como los eventos de otras entidades a ser ejecutados por éstos en el ambiente.

Metodología para la Captura de la base de conocimiento del agente

La siguiente metodología propone una forma de construir la base de conocimiento de la entidad, tomando como caso de estudio el vehículo:

1. Definir el área de percepción (AP) de la entidad. El AP permite limitar el área del ambiente que la entidad puede percibir en un instante de tiempo. Es utilizado para definir la longitud del área de aplicación de las tareas de soporte del PTD. Para el caso del vehículo, AP = 1 segmento. Esto permite reducir el número de eventos que la entidad tiene que manejar al iniciar el PTD. En el modelo, el espacio es descrito de manera continua, es decir, la posición de cada entidad en el STU es definida por un número de punto flotante (ver Fig. 2.11).

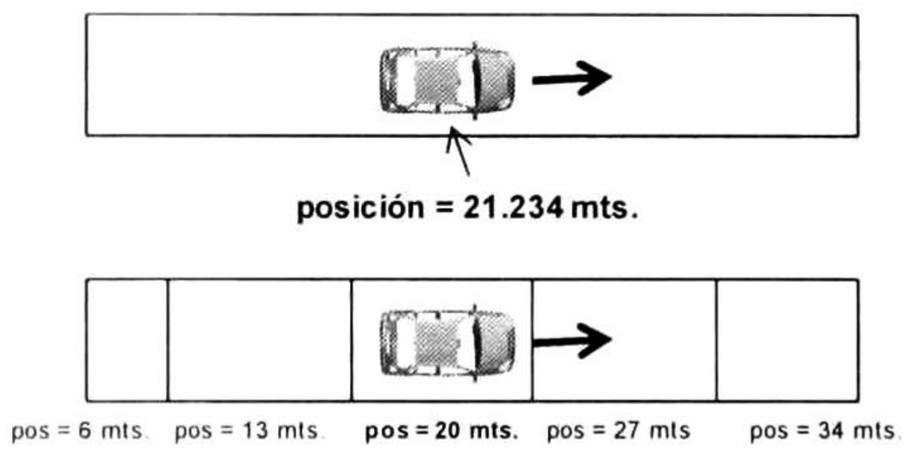


Fig. 2.11: Espacio continúo Vs. Espacio Discreto

2. Definir las ecuaciones que describen cada una de las actividades de la entidad. La dinámica del vehículo es simplificada con la siguiente ecuación involucrando solo la velocidad v como parámetro de la entidad, respecto a la distancia transcurrida $x(t) - x_0$, durante t-t_0.

$$x(t) = v * (t - t_0) + x_0$$

3. Definir las tareas de soporte en el PTD de la entidad. Durante el PTD, existen algunos puntos de decisión requieren involucrar los parámetros o atributos de la entidad ($entity_i$) y la ejecución de alguna tarea que permita establecer el valor de

ciertas variables de estado del sistema (búsqueda de obstáculo y validación del espacio para cambio de carril). Cada una de estas tareas al ser ejecutada sustituye los valores de las variables en las reglas de la base de conocimiento, permitiendo al agente tomar decisiones con respecto al estado de su ambiente. Definir las tareas de soporte en la toma de decisión de la entidad involucra el haber establecido previamente el AP del vehículo, las ecuaciones de movimiento y el comportamiento discreto del agente (*ActivityNET*$_{3,1}$). A continuación se describen cada una de las tareas de soporte del MTD del agente vehículo.

Búsqueda de obstáculo en el área de percepción del agente

Aquella entidad del STU que impida que alguna entidad móvil continúe avanzando libremente en un segmento es un obstáculo. Podemos clasificar dos tipos de obstáculos: aquellos generados por las PC (semáforos, señalamientos estáticos, etc.) y los generados por las RE (por ejemplo, un vehículo delante de otro).

En la Fig. 2.12 se puede observar un obstáculo del tipo RE. El vehículo identificado como *car*$_3$, se convertirá en obstáculo de aquel identificado como *car*$_2$ en cuanto el primero cambie de carril; y *car*$_1$ lo será pero para *car*$_3$.

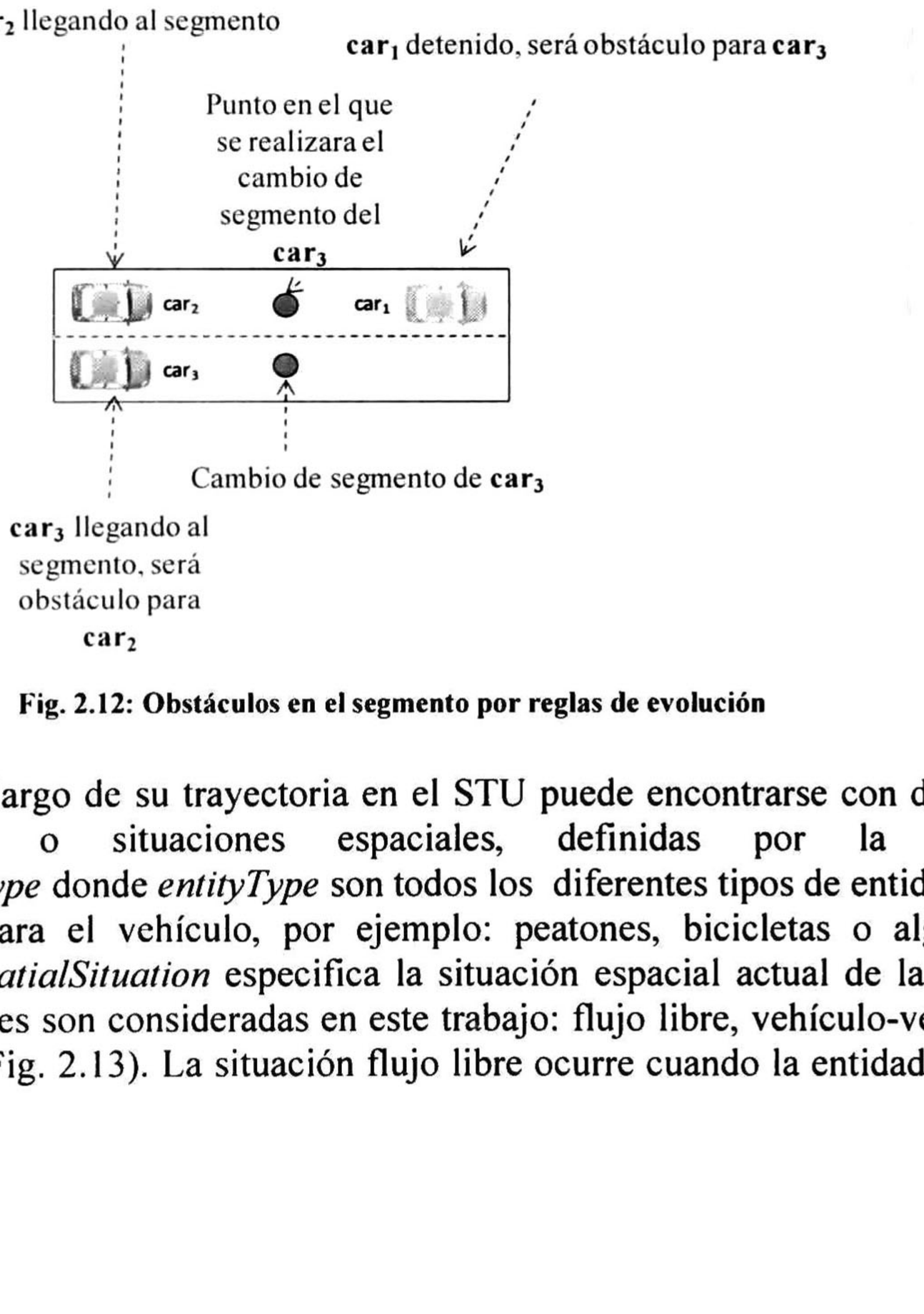
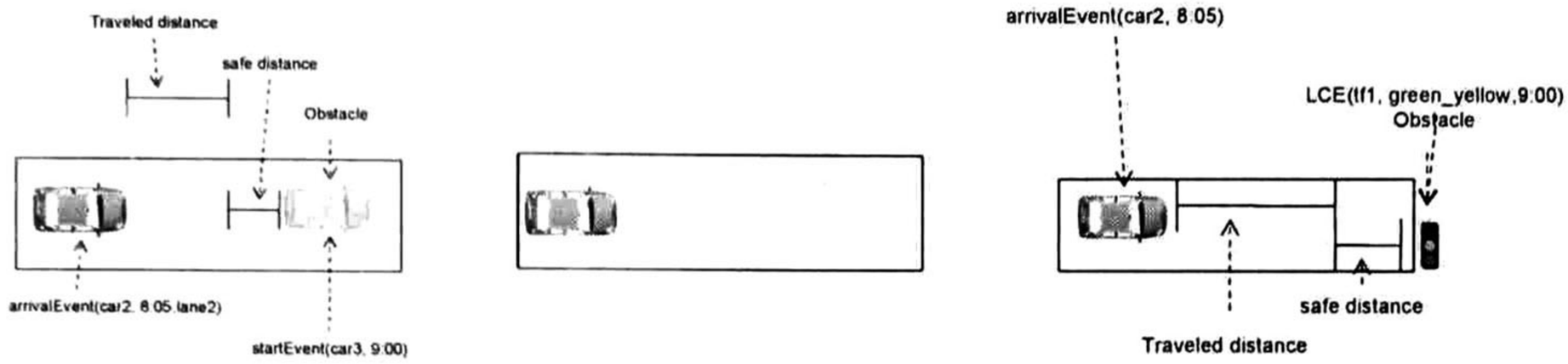


Fig. 2.12: Obstáculos en el segmento por reglas de evolución

El vehículo a lo largo de su trayectoria en el STU puede encontrarse con diferentes tipos de obstáculos o situaciones espaciales, definidas por la variable *spatialSituation=entityType* donde *entityType* son todos los diferentes tipos de entidades que pueden ser obstáculo para el vehículo, por ejemplo: peatones, bicicletas o algún otro vehículo. La variable *spatialSituation* especifica la situación espacial actual de la entidad. Tres situaciones espaciales son consideradas en este trabajo: flujo libre, vehículo-vehículo y vehículo-semáforo (ver Fig. 2.13). La situación flujo libre ocurre cuando la entidad no tiene

obstáculo alguno durante su trayecto en el segmento. La situación vehículo-vehículo representa ocurre cuando la entidad tiene otro vehículo al frente. La situación vehículo-semáforo describe a la entidad frente a un semáforo. Se puede incrementar el número de situaciones en el modelo, por ejemplo: vehículo-peatón, vehículo-señal_stop, vehículo-cruce_peatones, etc.



a) Situación Espacial vehículo-vehículo b) Situación Espacial Flujo Libre c) Situación Espacial Vehículo-Semáforo

Fig. 2.13: Diferentes situaciones espaciales

Una vez que el vehículo ha definido su situación espacial, es necesario encontrar el evento siguiente que representa al obstáculo en cuestión, asignándolo a la variable: e*ventObstacle* = $evt_i(time_i, position, segment, agent_i)$. Ésta representa el evento que la entidad observa en una situación espacial determinada. Este evento puede ser cualquiera de los eventos que puede generar la entidad (u objeto) obstáculo. Por ejemplo, para el caso vehículo-vehículo los eventos con los que se puede enfrentar son: *start*, *arrivalLink*, *warningStart;* y para la situación vehículo-semáforo: *redToGreen*, *greenToYellow*, *yellowToRed*.

También es necesario establecer si el vehículo llega al obstáculo *spatialSituation* antes o después que el evento e*ventObstacle* sea ejecutado, asignándolo a la variable *temporalRelation* el valor *before o after*. Ésta define la relación temporal existente entre la entidad y el evento del obstáculo (llega antes o después de ser ejecutado). Para definir el valor de la variable *temporalRelation* se utiliza la ecuación de velocidad media (Ecn. 1.2), que describe la velocidad de un vehículo respecto a la distancia transcurrida *d*, en un instante de tiempo *t*.

Utilizando la Ecn. 1.2 el vehículo podrá calcular si llega antes o después que sea ejecutado el evento del obstáculo. Los obstáculos generados porque la entidad obedece las PC (ver Fig. 2.14), se pueden clasificar en estáticos, es decir, su estado no cambia a lo largo del tiempo (alto total, cruce de peatones, etc.) y dinámicos, cuyo estado cambia a lo largo del tiempo (semáforos).

Para capturar el grado de negligencia existente en el STU real, se utilizan los parámetros *safeDistance* y *safechangeDistance*. El primero es utilizado para determinar la distancia que considera el vehículo como segura para colocarse detrás de un obstáculo, y el último es utilizado para verificar que tan lejos debe estar un vehículo de éste antes de hacer un cambio de carril. En ambos parámetros, entre menor sea su valor, el vehículo seguidor tendrá menor posibilidad de reaccionar a tiempo y tomar una decisión segura, incrementando la posibilidad de una colisión (ejecución de la ley de evolución 3, negligencia de la entidad).

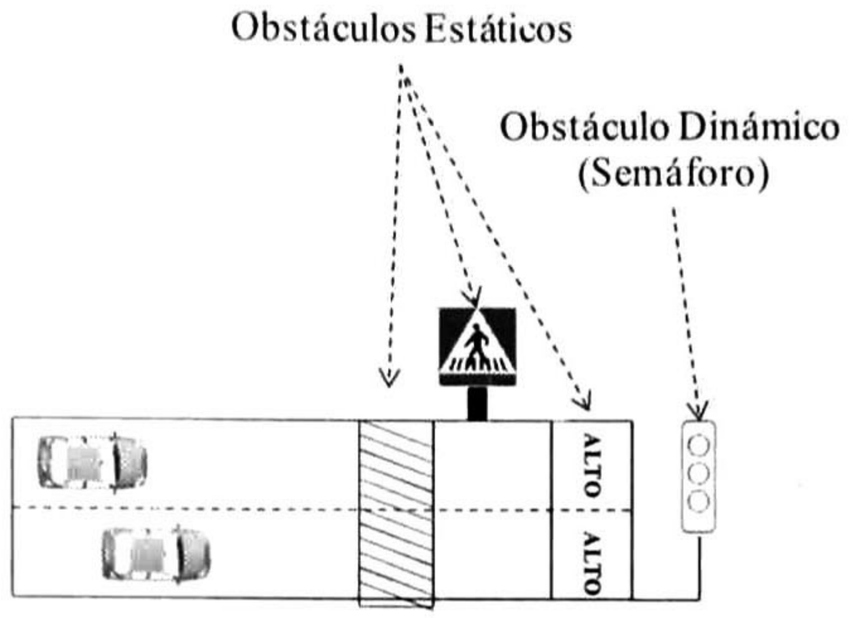


Fig. 2.14: Obstáculos en el Segmento por Políticas de Tránsito urbano

Escenario vehículo-semáforo.

La regla R_{1A} es aplicada cuando el vehículo calcula llegar hasta la posición de un semáforo cuando éste ya ha cambiado su luz a rojo. Es decir, *spatialSituation = trafficLight*, *event = yellowToRed* y *temporalRelation = after*. Entonces el vehículo genera los eventos *stop* y *start*.

La marca de tiempo del evento *stop* es igual al tiempo de llegada del vehículo (*arrivalTime)* a una distancia de seguridad del semáforo (*position = safeDistance*). Cuando la relación temporal del vehículo con el evento de cambio de luz ocurre antes del cambio, es decir *temporalRelation = before*, la regla R_{1B} es aplicada. Entonces el vehículo genera los eventos *leaveLink* y *arrivalLink*, para desplazarse longitudinalmente a otro segmento.

Las reglas definidas para las situaciones vehículo-vehículo y vehículo-semáforo son descritas respectivamente en la Tabla 2.8 y la Tabla 2.9

ID Regla	Condición	Acción
R_{1A}	**IF** *spatialSituation*[$actor_i$] == *trafficLight* **AND** *Event*[$actor_i$] == *yellowToRed* **AND** *temporalRelation*[$actor_i$, *yellowToRed*] == *After*	**THEN** ***stop***($actor_i$.*arrivalTime*, $actor_i$.*currentPosition* + $actor_i$.*traveledDistance*+ $actor_i$.*safeDistance*,$segment_i$, $actor_i$), ***start*** (*Event*[$actor_i$].*time* + actor$_i$.*getDelayToStart*,$segment_i$, $actor_i$)
R_{1B}	**IF** *spatialSituation*[$actor_i$] == *trafficLight* **AND** *Event*[$actor_i$] == *yellowToRed* **AND** *temporalRelation*[$actor_i$, *yellowToRed*] == *Before*	**THEN** ***leaveLink***($actor_i$.*arrivalTime*, $actor_i$.*currentPosition* + $actor_i$.*traveledDistance*,$segment_i$, $actor_i$), ***arrivalLink*** ($actor_i$.*arrivalTime*, 0.0,$segment_i$, $actor_i$)
R_{2A}	**IF** *spatialSituation*[$actor_i$] == *trafficLight* **AND** *Event*[$actor_i$] == *greenToYellow* **AND** *temporalRelation*[$actor_i$, *greenToYellow*] == *After*	**THEN** ***stop***($actor_i$.*arrivalTime*, $actor_i$.*currentPosition* + $actor_i$.*traveledDistance*+ $actor_i$.*safeDistance*,$segment_i$, $actor_i$), ***start*** (*Event*[$actor_i$].*time* + actor$_i$.*getDelayToStart*,$segment_i$, $actor_i$)
R_{2B}	**IF** *spatialSituation*[$actor_i$] == *trafficLight* **AND** *Event*[$actor_i$] == *greenToYellow* **AND** *temporalRelation*[$actor_i$, *greenToYellow*] == *Before*	**THEN** ***leaveLink***($actor_i$.*arrivalTime*, actor$_i$.*currentPosition* + $actor_i$.*traveledDistance*,$segment_i$, $actor_i$), ***arrivalLink*** ($actor_i$.*arrivalTime*, 0.0,$segment_i$, $actor_i$)

R_{3A}	**IF** *spatialSituation*[$actor_i$] == *trafficLight* **AND** *Event*[$actor_i$] == *redToGreen* **AND** *temporalRelation*[$actor_i$, *redToGreen*] == *After*	**THEN** ***leaveLink***($actor_i$.*arrivalTime*, $actor_i$.*currentPosition* + *traveledDistance*,$segment_i$, $actor_i$), ***arrivalLink*** ($actor_i$.*arrivalTime*, 0.0,$segment_i$, $actor_i$)
R_{3B}	**IF** *spatialSituation*[$actor_i$] == *trafficLight* **AND** *Event*[$actor_i$] == *redToGreen* **AND** *temporalRelation*[$actor_i$, *redToGreen*] == *Before*	**THEN** ***stop***($actor_i$.*arrivalTime*, $actor_i$.*currentPosition* + $actor_i$.*traveledDistance*+ $actor_i$.*safeDistance*,$segment_i$, $actor_i$), ***start*** (*Event*[$actor_i$].*time* + $actor_i$.*getDelayToStart*,$segment_i$, $actor_i$)

Tabla 2.8: Reglas if-then obtenidas con la metodología propuesta para el escenario vehículo-semáforo

Escenario vehículo-vehículo

La regla R_{4A} es aplicada cuando el vehículo calcula llegar hasta la posición de un vehículo que va al frente (vehículo líder), cuando el vehículo líder ya se ha desplazado longitudinalmente a otro segmento. Es decir, *spatialSituation = vehicle*, *event = leaveLink* y *temporalRelation = after*. Entonces el vehículo genera los eventos *warningStop* y *warningStart*.

La marca de tiempo del evento *warningStop* es igual al tiempo de llegada del vehículo (*arrivalTime)* a una distancia de seguridad de la posición del vehículo en la que salió del segmento. Cuando la relación temporal del vehículo con el evento de cambio de segmento (leaveLink) ocurre antes del cambio, es decir *temporalRelation = before*, la regla R_{4B} es aplicada. Entonces el vehículo genera los eventos *warningStop* y *warningStart*, para evaluar la posibilidad de un paro total.

ID Regla	Condición	Acción
R_{4A}	**IF** *spatialSituation*[$actor_i$] == *vehicle* **AND** *Event*[$actor_i$] == *leaveLink* **AND** *temporalRelation*[$actor_i$, *leaveLink*] == *After*	**THEN** ***warningStop***($actor_i$.*arrivalTime*, $actor_i$.*currentPosition* + $actor_i$.*traveledDistance*,$segment_i$, $actor_i$), ***warningStart*** ($actor_i$.*currentPosition* + $actor_i$.*traveledDistance* , $actor_i$.*arrivalTime*,$segment_i$, $actor_i$)
R_{4B}	**IF** *spatialSituation*[$actor_i$] == *vehicle* **AND** *Event*[$actor_i$] == *leaveLink* **AND** *temporalRelation*[$actor_i$, *leaveLink*] == *Before*	**THEN** ***warningStop***($actor_i$.*arrivalTime*, $actor_i$.*currentPosition* + $actor_i$.*traveledDistance*+ $actor_i$.*safeDistanc*,$segment_i$, $actor_i$), ***warningStart*** (*Event*[$actor_i$].*time* + $actor_i$.*getDelayToStart*,$segment_i$, $actor_i$)
R_{5A}	**IF** *spatialSituation*[$actor_i$] == *vehicle* **AND** *Event*[$actor_i$] == *start* **AND** *temporalRelation*[$actor_i$, *start*] == *After*	**THEN** ***warningStop***($actor_i$.*arrivalTime*, $actor_i$.*currentPosition* + $actor_i$.*traveledDistance*+ $actor_i$.*safeDistance*,$segment_i$, $actor_i$), ***warningStart*** (*Event*[$actor_i$].*time* + $actor_i$.*getDelayToStart*,$segment_i$, $actor_i$)
R_{5B}	**IF** *spatialSituation*[$actor_i$] == *vehicle* **AND** *Event*[$actor_i$] == *start* **AND** *temporalRelation*[$actor_i$, *start*] == *Before*	**THEN** ***stop***($actor_i$.*arrivalTime*, $actor_i$.*currentPosition* + $actor_i$.*traveledDistance*+ $actor_i$.*safeDistance*,$segment_i$, $actor_i$), ***start*** (*Event*[$actor_i$].*time* + $actor_i$.*getDelayToStart*,$segment_i$, $actor_i$)

R_{6A}	**IF** *spatialSituation*[$actor_1$] == *vehicle* **AND** *Event*[$actor_1$] == *warningStart* **AND** *temporalRelation*[$actor_1$, *warningStart*] == *After*	**THEN** ***warningStop***($actor_1$.*arrivalTime*, $actor_1$.*currentPosition* + $actor_1$.*traveledDistance*+ $actor_1$.*safeDistance*,$segment_n$, $actor_1$), ***warningStart*** (*Event*[$actor_1$].*time* + $actor_1$.*getDelayToStart*,$segment_n$, $actor_1$)
R_{6B}	**IF** *spatialSituation*[$actor_1$] == *vehicle* **AND** *Event*[$actor_1$] == *warningStart* **AND** *temporalRelation*[$actor_1$, *warningStart*] == *Before*	**THEN** ***warningStop***($actor_1$.*arrivalTime*, $actor_1$.*currentPosition* + $actor_1$.*traveledDistance*+ $actor_1$.*safeDistance*,$segment_n$, $actor_1$), ***warningStart*** (*Event*[$actor_1$].*time* + $actor_1$.*getDelayToStart*,$segment_n$, $actor_1$)

Tabla 2.9: Reglas *if-then* obtenidas con la metodología propuesta para el escenario vehículo-vehículo

Validación de espacio para cambio de segmento.

Esta habilidad es utilizada una vez que el vehículo ya ha tomado la decisión de cambiar de segmento, y es necesario validar la existencia de espacio disponible. Por lo tanto define el valor de las variables *safeChange* y *conflictActor*. Utilizando el algoritmo 1 con el ejemplo de la Fig. 2.12, donde car_3 se encuentra en un proceso de toma de decisión, las variables quedarían asignadas como sigue:

Cuando la variable *safeChange* = true, si la distancia entre el cambio de segmento y la posición de car_2 es segura (dependerá del parámetro safechangeDistance de car_3); en caso contrario *safeChange* = true.

Cuando la variable *conflictActor* = car_2, el car_3 colocará el evento *resolveConflict* para car_2 informándole de la decisión del nuevo evento de cambio de segmento, de manera que car_2 re-evalúe su último evento futuro.

El evento *inConflict*, permite que dos entidades móviles se comuniquen directamente. Este evento hará que el vehículo en conflicto actualice su estado hasta el instante en el que el vehículo que generó el conflicto tomó la decisión de cambiar de carril.

Esta tarea de soporte es utilizada una vez que el vehículo ya ha tomado la decisión de cambiar de segmento y es necesario validar la existencia de espacio disponible por lo tanto define el valor de las variables *safeChange* y *conflictActor*. La implementación de esta tarea de soporte es descrita por los algoritmos 4.6 y 4.7.

Algoritmo 4.6 ¿Existe espacio disponible en la posición po*sibleClPosition* en el instante *posibleClTime?*

```
1:  Func: possibleGap(posibleClPosition,posibleClTime:double)
2   set carTemp = carAtObjectiveSegment()
3:  set carPosition = carsPosition.get(carTemp.id)
4:  // Obtiene la imagen de la posición del vehículo al instante del cambio de segmento
5   set shootPosition = carTemp.positionAt(posibleClTime)
6:  if (shootPosition = posibleClPosition) then
7:          safeChange[cari] = false
8:          conflictActor[cari] = carTemp
9:          return
10: else
11:         if (shootPosition < posibleClPosition ) then
12:             if(carTemp.typeSeq = CE)then
13:                 gap.status = false
14                  gap carConflict = carTemp
```

```
15                                  gap.distanceConflict = posibleClPosition – shootPosition
16.                                         return gap
17·                                 end if
18                          end if
19:             end if
20              set i = i+1
21      end while
22      return gap
```

Algoritmo 4.7 Encuentra la posición de un vehículo en el instante, atributos del vehículo: *carPosition, stopTime, typeSeq, begEvTime, stopDeltaP,evTime, endEvTime.*

```
1       Func positionAt(instant double).double
2       if (typeSeq = SE) then
3               if(evTime >= stopTime) then
4                       return carPosition
5               else
6                       if(instant < stopTime)then
7.                              set deltaP = (instant – begEvTime) * velocity
8:                              set posInstant = carPosition + deltaP
9                               return posInstant
10:                     else
11.                             return stopDeltaP + carPosition
12                      end if
13:             end if
14      else
15              if (instant < endEvTime) then
16:                     set deltaP = (instant-begEvTime) * velocity
17·                     set posInstant = carPosition + deltaP
18:                     return posInstant
19:             else
20·                     if (typeSeq = CLE) then
21                              return changeLaneDeltaP + carPosition
22:                     else
23:                             return crossDeltaP + carPosition
24                      end if
25:             endif
26:     end if
```

Frenado seguro

Esta habilidad tiene que asignar a la variable *safeBreak* el valor de verdadero o falso. Si el vehículo *i* alcanza a frenar con su parámetro de velocidad, antes de colisionar con el vehículo que se encuentra delante de él.

2.3. Modelo del Sistema de Tráfico Urbano basado en Agentes

El modelo n-LNS de STU presentado en la sección anterior, podría implementarse en algún simulador de RP multi-nivel, sin embargo hasta el momento no existe alguno que asocie el mecanismo de toma de decisiones a la RP. Utilizaremos la jerarquía del modelo n-LNS de STU (ver Fig. 2.15), para hacerlo corresponder con la definición de SMA. Es decir,

el primer nivel que describe la estructura de la red vial, hacerlo corresponder como el ambiente del SMA; el segundo nivel que describe la estructura de la entidad y el tercer nivel en el que se especifican las actividades de la entidad asociado a un MTD, lo hacemos corresponder con la definición de agente.

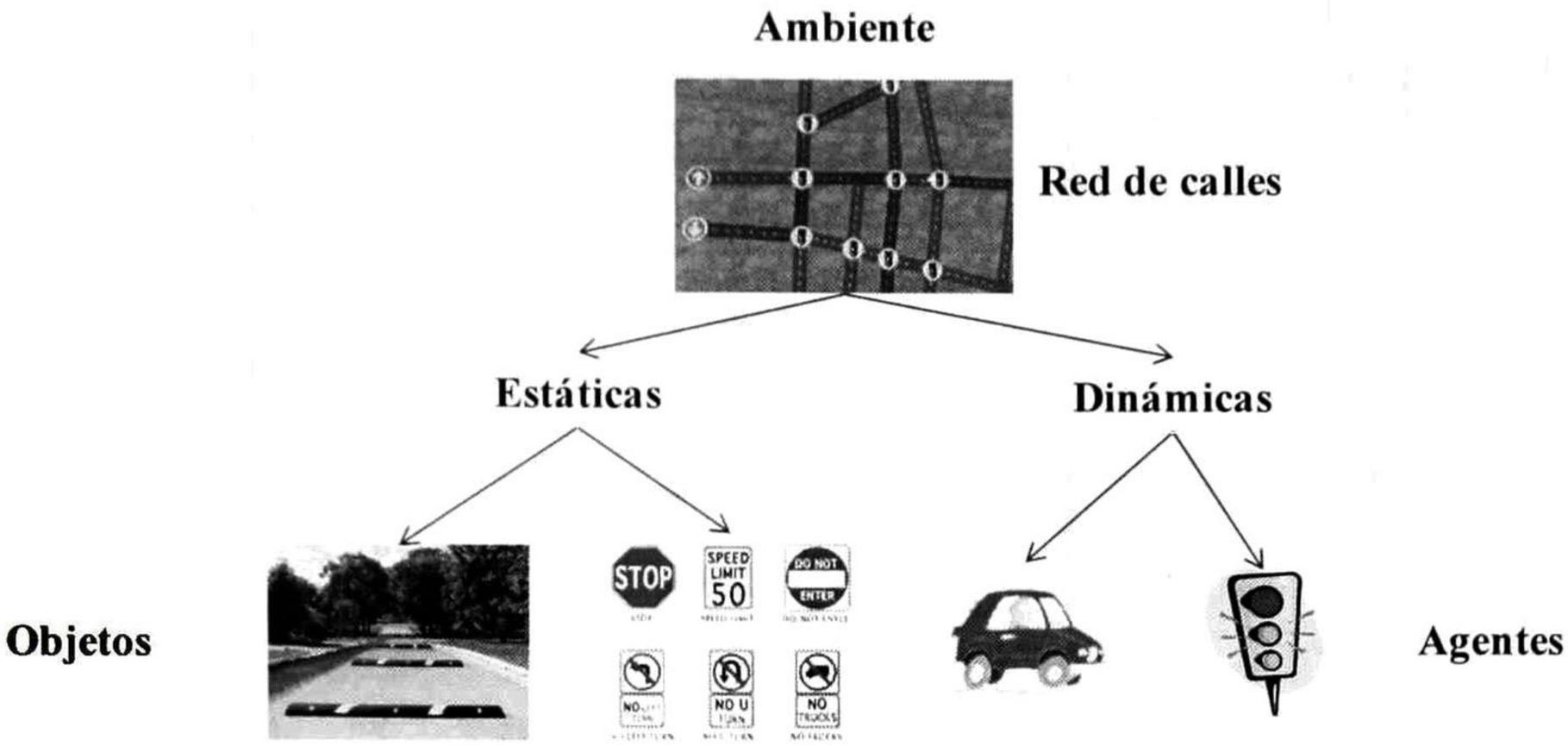


Fig. 2.15: Clasificación de los Elementos del Sistema de Tráfico Urbano

Entonces, podemos definir un agente con la 3-tupla siguiente:

Definición 2.13. Un agente es definido por $\boldsymbol{agent_i}$ = ($EntityNET_2$, $entity_i$, MTD_i), donde:

- $EntityNET_2$ es la descripción del comportamiento de la entidad para desplazarse en el ambiente.
- $entity_i$ es la descripción particular de una entidad.
- MTD_i es el mecanismo de toma de decisión asociada a la entidad.

Se puede definir el modelo de un STU basado en multi-agentes como sigue:

Definición 2.14. El modelo de SMA del STU es una 3-tupla, definida por **stubma** = (*3-LNS*, *S*, *A*) donde:

- *3-LNS* es la definición de ambiente y entidades usando el formalismo de RP de tres niveles 3-LNS (ver Fig. 2.16). El primer nivel del modelo es llamado $EnvironmentNET_1$. Cada lugar de esta red representa una unidad espacial de modelado básica llamada segmento. El segundo nivel llamado $EntityNET_2$, provee de la estructura necesaria para que el agente pueda desplazarse en la red del primer nivel, haciendo uso de la red de tercer nivel, $ActivityNET_3$, que describe las actividades que puede ejecutar el agente en el modelo

- *S* representa el conjunto de segmentos del modelo n-LNS del STU.
- *A* representa el conjunto de entidades en el STU, tal que $A=\{entity_i\}$.

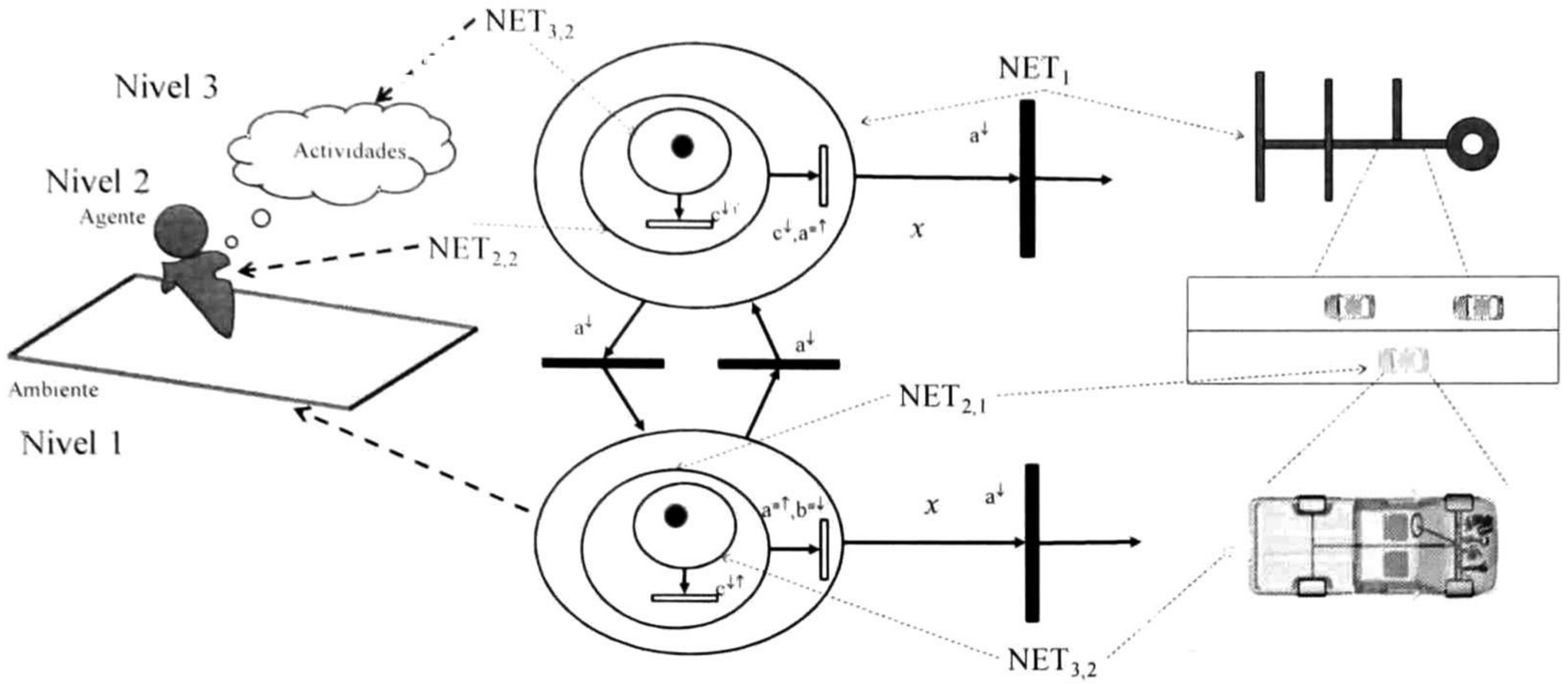


Fig. 2.16: Modelo de STU bajo el Enfoque de Sistemas Multi-Agente

2.4. Metodología para la Captura del Comportamiento Discreto de los agentes

El comportamiento de las entidades deberá "discretizarse" de manera que pueda ser capturado por las redes de nivel 3. A continuación se describe la metodología para capturar el comportamiento de la entidad en el modelo n-LNS del STU. Se usará como caso de estudio el vehículo (en este trabajo se considera como una sola entidad al conductor y al vehículo):

1. Establecer la variable (cuantitativa) que permita verificar el comportamiento de la entidad con respecto al tiempo. La variable seleccionada deberá de proveer la información suficiente para poder verificar todos los posibles cambios de actividad de la entidad. En el caso del vehículo, es utilizada la variable velocidad.
2. Definir las actividades que la entidad puede realizar en el STU e identificar aquellos eventos en los que se puede decidir el inicio o término de dichas actividades. Estos eventos deberán ser seleccionados de manera que no sea necesario hacer referencia a algún otro evento intermedio y pueda ser definido el comportamiento de la entidad. Debe tenerse en cuenta que del número de actividades definidas dependerá el nivel de granularidad del modelo. Comúnmente el comportamiento del vehículo es capturado por los modelos de seguimiento de vehículos. Esto es, si el vehículo líder desempeña alguna maniobra de manejo, el vehículo seguidor, responderá con cambios en su velocidad. Las actividades que el vehículo puede realizar son arrancar, dejar segmento, llegar a otro segmento, realizar cambio de carril, llegar a un carril y parar. Los eventos que inician dichas actividades están descritos en la Tabla 2.10.

Usando estas actividades y eventos puede describirse el comportamiento de un vehículo graficando el cambio de velocidad con respecto al tiempo (ver Fig. 2.17): el vehículo llega al segmento, avanza por un instante y después se detiene; se mantiene en este estado por un tiempo hasta que arranca y entonces el vehículo avanza y por último deja el segmento para trasladarse a algún otro.

Evento	Descripción
initArrivalSegment	Arribo de un vehículo a un segmento en un movimiento secuencial
initStart	Inicia a avanzar después de un reposo total
initStop	Inicia reposo total
initLeaveSegment	El vehículo deja el segmento en el que venía viajando
initChangeLane	El vehículo cambia de carril (cambio de segmento transversal)
initArrivalLane	Llegada de un vehículo a un segmento en un movimiento de cambio de carril (movimiento transversal)
initSenseAdvancing	Verificar FOV del vehículo para tomar decisiones, mientras el vehículo sigue avanzando
initSenseStopped	Verificar FOV del vehículo para tomar decisiones, mientras el vehículo sigue detenido

Tabla 2.10: Descripción de eventos del vehículo

3. Identificar aquellos sucesos que mantengan o cambien (*initStart, initStop y initChangeLane*) la actividad de la entidad, así como los atributos modificados por dichos sucesos (velocidad, posición, aceleración, etc.). Éstos son capturados como eventos. Algunos de estos eventos también pueden describir el inicio de un PTD por parte de la entidad (*initSenseAdvancing, initSenseStopped, arrival, arrivalSegment y arrivalChangeSegment*). Se genera la red $ProcessNet_{3,1}$, los eventos que hacen que la entidad cambie de actividad etiquetarán las transiciones que cambien el estado de la red. Los eventos restantes serán etiquetas de transiciones que mantienen a la entidad en el mismo estado.
4. Una vez que se han obtenido las actividades y eventos, se construyen las redes de nivel 3 $ActivityNET_3$ del vehículo. Cada actividad será capturada como una red y se sincronizará con la red $ProcessNET_{3,1}$.

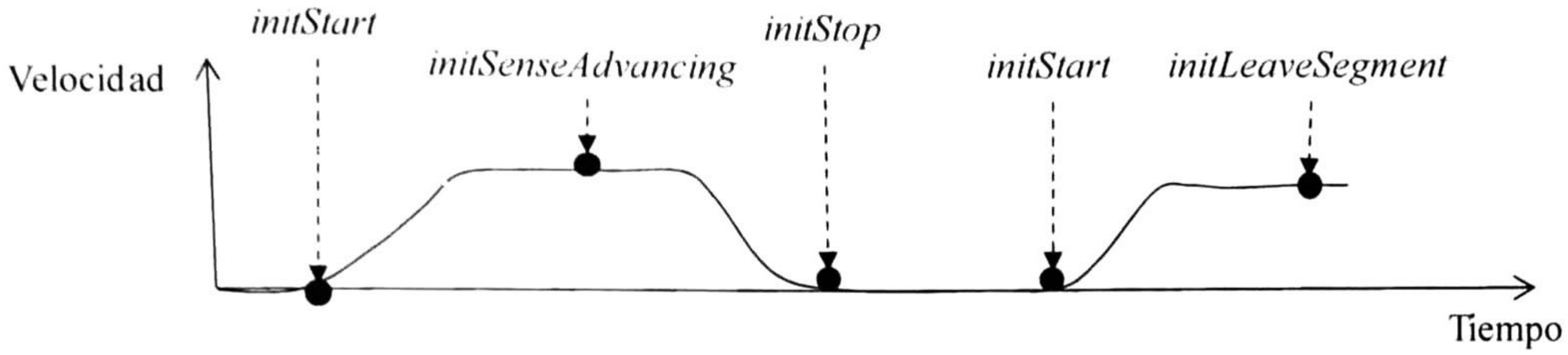


Fig. 2.17: Ejemplo del comportamiento discreto del vehículo

2.5. Discusión

En este capítulo se presentó una metodología para el modelado de STU. El modelo se obtiene utilizando el formalismo de RP multi-nivel (3-LNS), para capturar el comportamiento de las entidades dinámicas, así como sus relaciones en el STU, logrando simplicidad y modularidad en el diseño. Además, la descripción del modelo del comportamiento de la entidad con las redes *ActivityNET*$_3$ permite seleccionar el nivel de detalle microscópico deseado del comportamiento del agente. Esto permite integrar en el modelo diferentes comportamientos de entidades dinámicas del STU, tales como: vehículos, semáforos, peatones, bicicletas, etc. propiedad requerida en los simuladores actuales de STI. Además, se presenta una correspondencia entre el modelo n-LNS y los SMA, que permite la implementación del modelo en algún lenguaje de programación.

CAPITULO 3

ARQUITECTURA DISTRIBUIDA PARA LA SIMULACIÓN DE SISTEMAS DE TRÁFICO URBANO

En este capítulo se presenta una estrategia para distribuir la simulación del modelo del STU basado en n-LNS y multi-agente. En primera instancia se presenta la forma de distribuir los eventos en una lista centralizada; este enfoque nos permitirá ejecutar de manera concurrente eventos de diferentes entidades en el mismo paso de simulación. Siguiendo ciertas reglas para conservar una causalidad válida en el sistema, se propone una arquitectura distribuida, así como un mecanismo para la ejecución distribuida del modelo de STU.

3.1. Arquitectura de simulación

Aunque el planteamiento de la simulación del STU bajo el paradigma de SMA es relativamente simple, existen algunos problemas de implementación que deben tomarse en cuenta. El primero es cómo mantener relaciones de causalidad válidas cuando se desea realizar simulaciones concurrentes; el segundo se debe al elevado número de recursos computacionales requeridos por este enfoque (ya sea por la granularidad del modelo o el nivel de comunicación entre los agentes).

Con el fin de manejar estos problemas, se propone la arquitectura distribuida y concurrente mostrada en la Fig. 3.1. En esta arquitectura es usado el concepto de proceso. Un proceso es aquel que contiene su propio flujo de control, a diferencia de las estructuras de datos. Éste puede contener referencia a otros procesos o estructuras de datos.

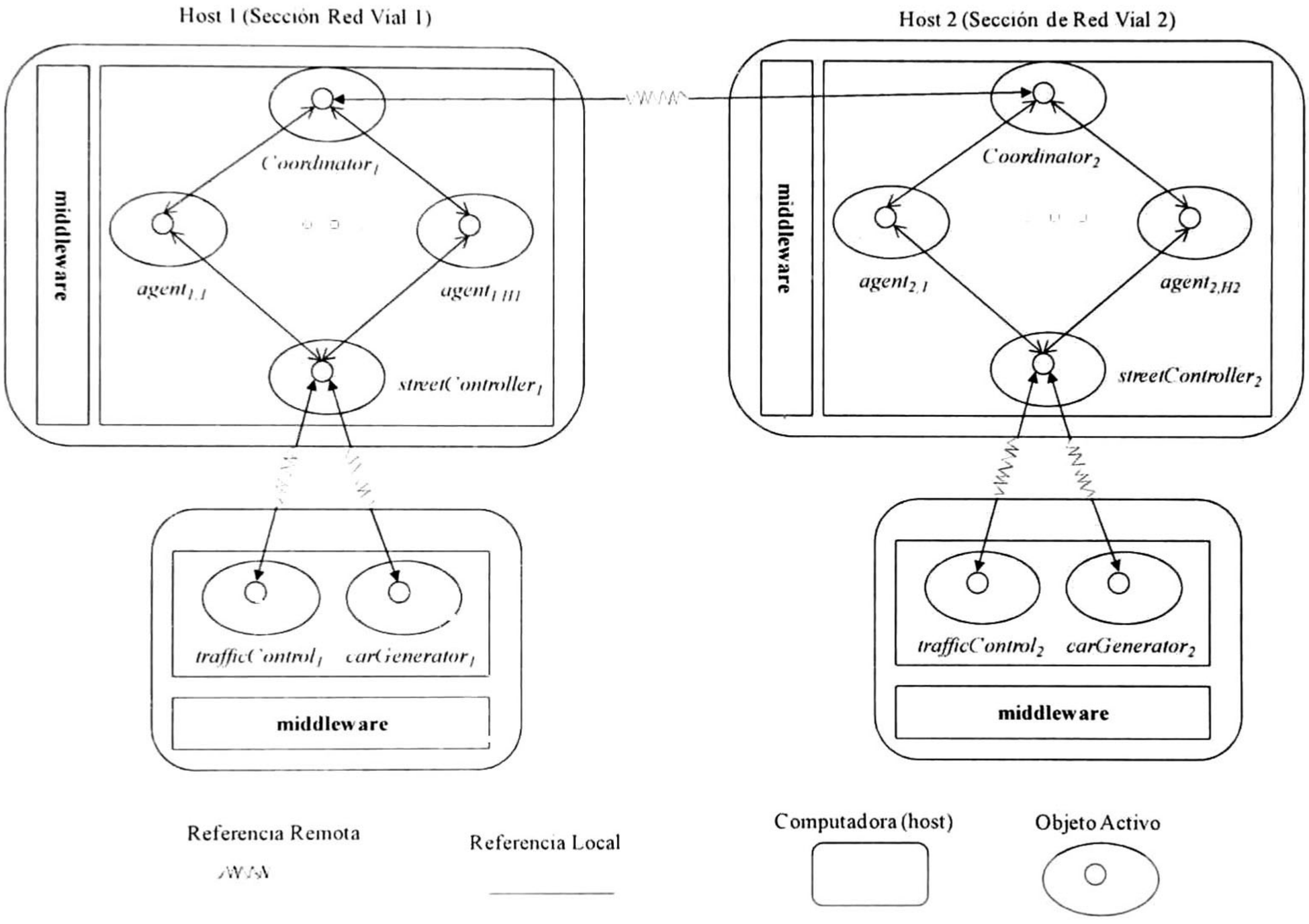


Fig. 3.1: Arquitectura concurrente y distribuida del simulador de tráfico urbano

Los procesos son difundidos en computadoras distribuidas. Una estructura de red vial puede ser dividida en secciones para su simulación. Cada sección es asignada a una computadora (host). Cada host tiene su propio generador de vehículos y control de tráfico, los eventos generados por cada uno de éstos es recibido por el proceso *StreetController*. El proceso generador de vehículos (*carGenerator*) utiliza una base de datos de información demográfica para estar generando los vehículos durante la simulación. El proceso *trafficControl* se encarga de generar los cambios de estados de los semáforos en la sección de red vial asignada. El proceso $StreetController_y$ se encarga de ejecutar concurrentemente los agentes $agent_{y,1}$ al agente $agent_{y,Hx}$, tal que y es el id del host en el que se ejecuta el agente y x es el número de agentes que concurrentemente pueden ser ejecutados en un instante (pertenecen al conjunto *History* que será descrito más adelante), el número de agentes que pueden ser ejecutados de

manera concurrente es limitado con el fin de conservar relaciones de causalidad válidas durante la simulación.

El proceso identificado como $Coordinator_y$, recibe los mensajes de solicitud de los agentes *agent* que deseen desplazarse a un segmento que se encuentra en otra sección de red vial. El coordinador consulta sobre la disponibilidad de espacio en el segmento (al cual el agente desea desplazarse) al coordinador del host destino, ésta información es enviada al agente que la solicitó para que pueda definir su acción siguiente en su proceso de toma de decisión.

A continuación se presentan en detalle los componentes de la arquitectura concurrente y distribuida de simulación del STU, así como las estructuras de datos que permiten capturar y mantener relaciones de causalidad válidas en el STU, éstas a su vez permitirán proponer una estrategia de ejecución concurrente y el diseño de una estrategia de ejecución distribuida del modelo de simulación presentado en el capítulo anterior.

3.2. Representación de las relaciones de causalidad del sistema de tráfico urbano

En los modelos en 3-LNS del capítulo anterior, las transiciones pueden ejecutarse de manera concurrente, sin embargo esto no debe ocurrir para el modelo del STU, ya que es un sistema causal. Aunque las RP capturan causalidad, en el modelo el orden de ejecución de los eventos no es capturado, ya que el PTD no es implementado en el modelo, así pues no podemos asegurar un correcto orden de causalidad en el modelo.

Para capturar la causalidad del STU se describe la estructura de eventos futuros (E^2F). La E^2F es un modelo de procesos de cómputo. En ésta se representa a un proceso como un conjunto de ocurrencias de eventos con relaciones que expresan como los eventos dependen causalmente de otros. La E^2F permite capturar y describir las leyes de evolución, las cuales son declaradas en la base de conocimiento ($knowledgeBase_i$) de la entidad. Un evento o el disparo de la transición t_1 de las redes tipo $ActivityNET_3$, es descrito por:

$$evt_i(time_i, w_i, s_i, e_i)$$

Donde:

- $time_i$ es el instante de ejecución del evento, tal que $time_i \in \aleph$.
- w_i es la posición de la ejecución del evento en el segmento, tal que $w_i \in \Re^+$
- s_i es el segmento en el que se ejecutará el evento, tal que $s_i \in S$.
- e_i es la entidad ($entitiy_i$) que colocó el evento.

Los elementos de la E^2F están relacionados por $\geq$ y Γ. Con el fin de describir dichas relaciones, la siguiente notación es definida:

$evt_i.s$ = segmento en el que se ejecutará el evento.
$evt_i.time$ = marca de tiempo del evento.
$evt_i.e$ = entidad que generó el evento.

La siguiente relación de dependencia Γ representa la relación existente entre la $entity_j$ que es observada por la $entity_i$, para realizar su toma de decisión, esto es, la $entity_j$ es un obstáculo para la $entity_i$:

Relación 3.1: Dependencia:

$\Gamma = \{ (evt_i, evt_j) \in E^2F \times E^2\mathrm{F} \mid evt_i.s = evt_j.s$ y $evt_i.e$ fue observado por $evt_j.e$ en su proceso de toma de decisión)$\}$

La siguiente relación permite representar la relación de causalidad de los eventos generados por las entidades, esto es, evt_i es ejecutado antes que evt_j y se ejecutará en el mismo segmento:

Relación 3.2: Causalidad:

$$\leq = \{ (evt_i, evt_j) \in E^2F \times E^2\mathrm{F} \mid evt_i.t \leq evt_j.time \text{ y } evt_i.s = evt_j.s \ \}$$

Las relaciones entre los eventos colocados por las entidades que participan en el escenario de la Fig. 3.2a, son mostradas en la Fig. 3.2b. Los eventos de la E^2F, son etiquetados como evt_i. Las flechas en dirección descendente describen la relación $\leq$ entre los eventos, por ejemplo, el evento evt_1 ocurrirá antes que el evento evt_2 ocurra. Las flechas en dirección ascendente describen la relación Γ entre los eventos, por ejemplo, el evento evt_3 es observado por la entidad del evento evt_2 como un obstáculo. Usando estas relaciones algunos axiomas pueden ser definidos.

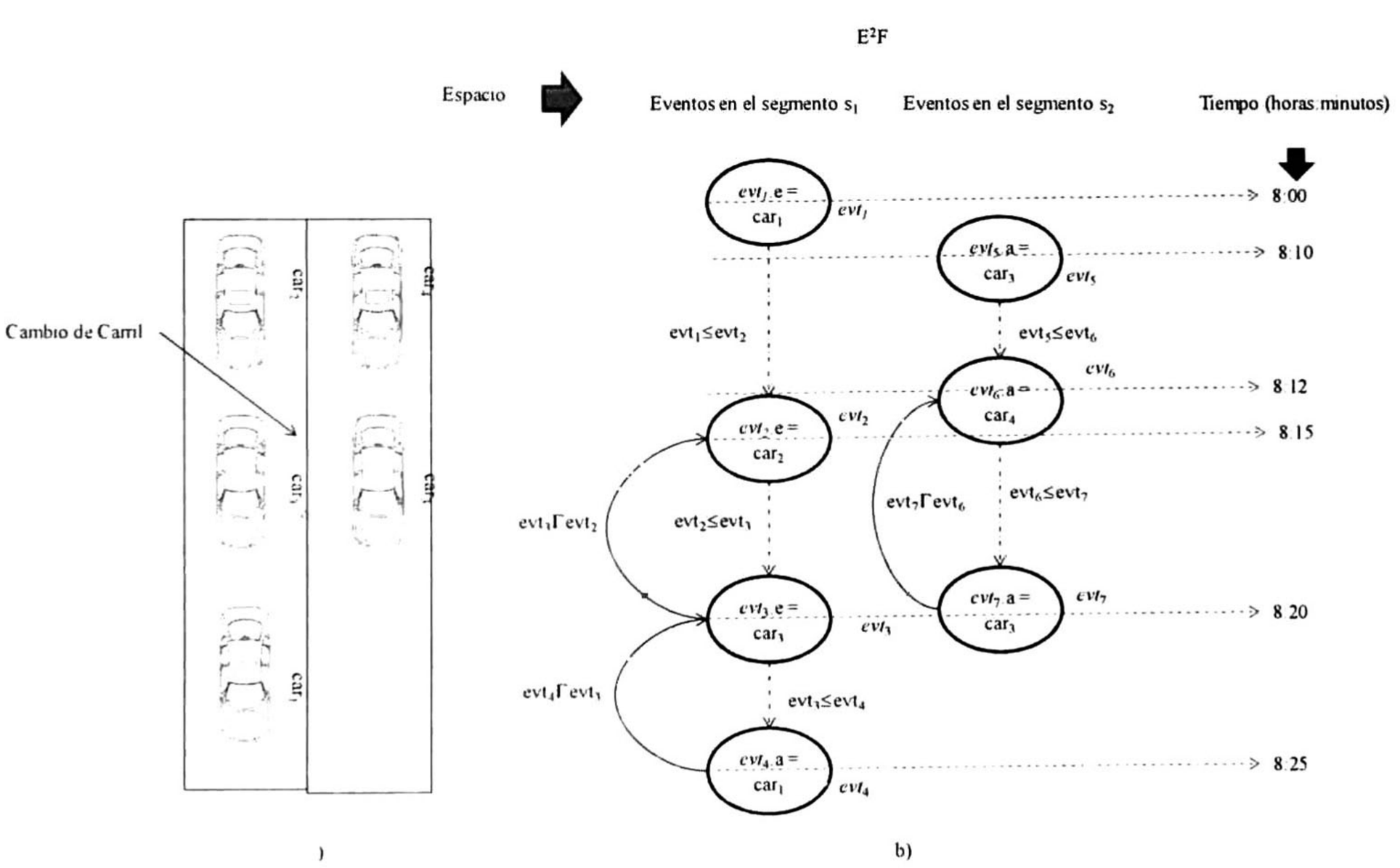


Fig. 3.2. Estructura espacio-tiempo de la secuencia de eventos

Axioma 3.1: Existencia de Entidades. Si un segmento s_i contiene alguna entidad, entonces deberá existir al menos un evento a ser ejecutado en la estructura E^2F para cada entidad en el segmento s_i.

$\forall x$(x es una entidad en el modelo $\Rightarrow \exists\ evt_i \in E^2F,\ evt_i.e = x$)

El siguiente axioma captura una de las leyes de evolución del STU. El axioma describe la restricción física del vehículo al no poder pasar sobre el vehículo que tiene adelante.

Axioma 3.2: Secuencialidad

Si $\exists x,y,z \in E^2F$ y $(x,y)\in\Gamma$ y $(x,z)\in\Gamma \Rightarrow y = z \vee x.e$ es un semáforo.

3.3. Estrategias de ejecución del modelo de STU

Los agentes y sub-sistemas (e.g. control de tráfico, información demográfica, etc.) del modelo del STU, son la única fuente de eventos. Estos eventos son usados como medio de comunicación e interacción entre los agentes. Se requiere un mecanismo (máquina de simulación) que ejecute estos eventos. Además, por ser el modelo basado en SMA se requiere otro mecanismo que ejecute el proceso de toma de decisión de cada agente (máquina de toma de decisión). Este enfoque permite describir la ejecución de la simulación en dos niveles de abstracción: La máquina de simulación y la máquina de toma de decisión (ver Fig. 3.3). En este capítulo estamos interesados en el mecanismo de ejecución de los eventos generados por los agentes y que es implementado por el objeto activo *StreetController* de la arquitectura distribuida presentada en la Fig. 3.1.

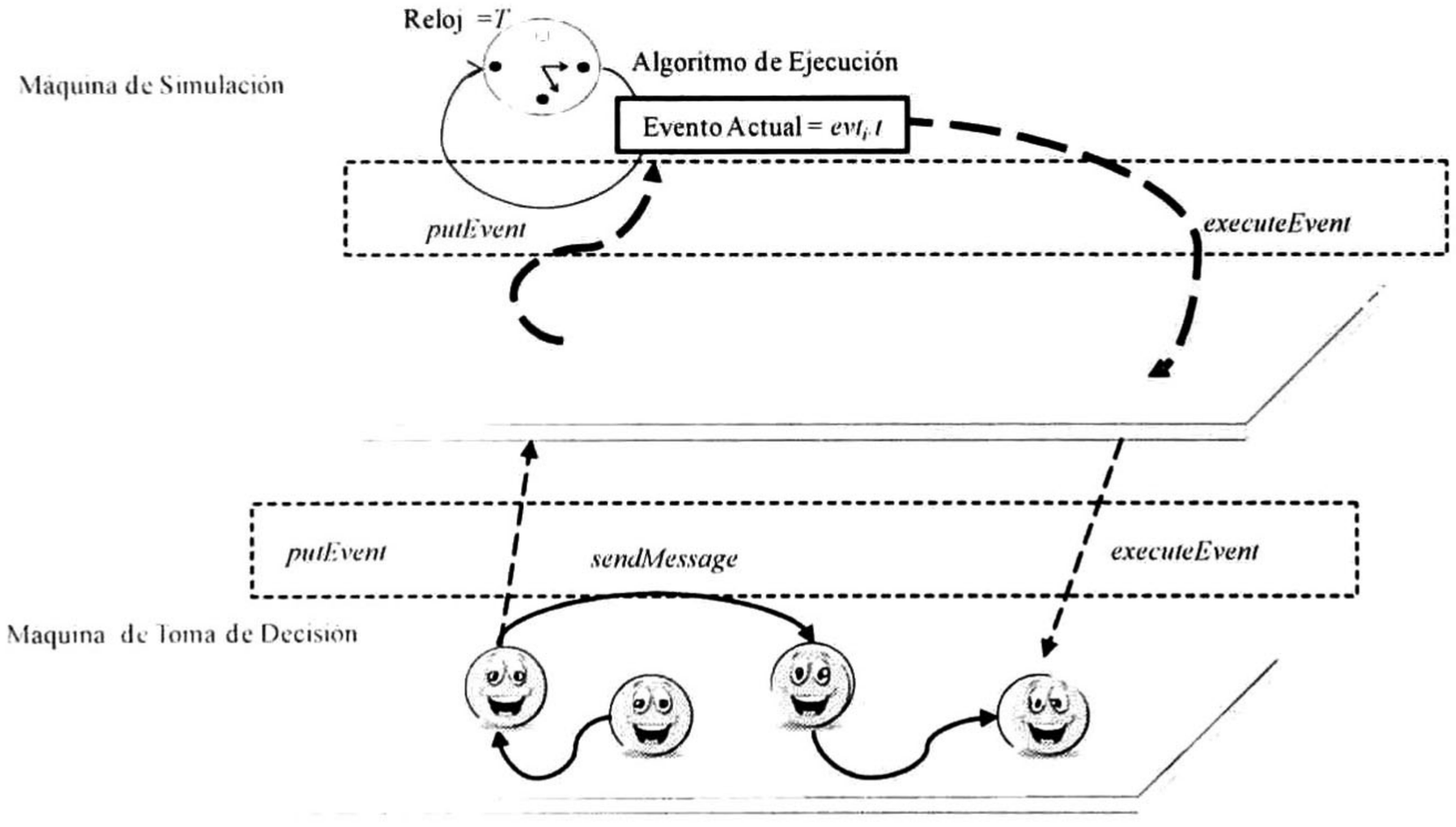


Fig. 3.3: Máquina de Simulación y máquina de toma de decisión

Los eventos son creados dinámicamente y pasan por dos fases: espera y ejecución. Una de las tareas más importantes de la máquina de simulación es mantener una relación de causalidad válida entre los eventos; esto es, aunque los eventos sean creados dinámicamente, ningún evento con $evt_i.time < T$ puede ejecutarse en el instante de simulación T.

Cuando un evento es creado, este puede ser ejecutado directamente (si $evt_i.time = T$) o puede ser colocado en la lista de eventos futuros (LEF) (si $evt_i.time > T$). La LEF es esencialmente una cola de prioridad que permite controlar el flujo de simulación de los eventos con mayor prioridad (menor marca de tiempo) y menor prioridad (mayor

marca de tiempo). Estos eventos deben ser procesados en orden temporal no decreciente.

En caso de existir varios eventos con la misma marca de tiempo, éstos deberán ser procesados en el orden que fueron insertados en la LEF Un evento contenido en la LEF cambia a la fase de ejecución cuando el reloj de simulación alcanza su marca de tiempo. La máquina de simulación avanza el reloj de simulación a la marca de tiempo del evento que sacó de la LEF (Fig. 3.4).

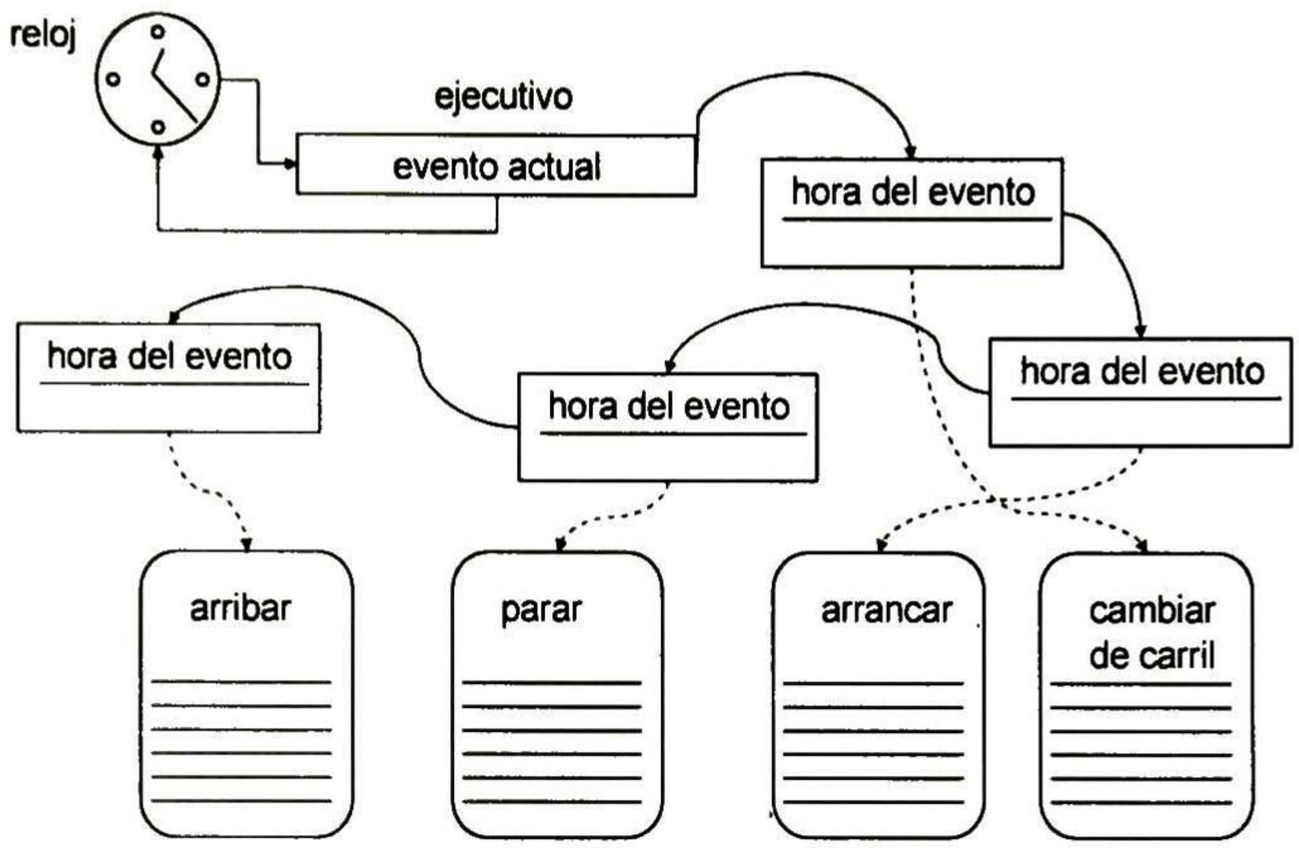


Fig. 3.4: Mecanismo de simulación secuencial

Una máquina de simulación ejecutando el mecanismo secuencial anterior, no puede ejecutar dos eventos, evt_i y evt_j, con la misma marca de tiempo (como se esperaría ocurriera en el mundo real), ya que no se puede asegurar una relación de causalidad válida entre estos eventos, es decir, no podríamos asegurar que no ocurriera la ejecución del evento evt_i, en el caso que éste dependiera de la ejecución del evento evt_j (o viceversa). Supongamos el caso de un vehículo detrás de otro, no podríamos ejecutar ambos eventos ya que uno de ellos (el que va detrás) no sabría cual es el resultado de la toma de decisión del vehículo que va al frente, al ser ejecutado su evento asociado.

A continuación se presenta una estrategia de ejecución secuencial, que posteriormente será usada como base para la ejecución concurrente y distribuida de los eventos del STU.

3.3.1. Ejecución secuencial con listas de eventos distribuidos

La estructura E^2F describe las restricciones en la ejecución de los eventos generados durante la simulación. Una forma de capturar esta restricción es asignando a cada segmento s_i una fragmento de la E^2F, llamado lista de eventos futuros (LEF). En el capítulo anterior se definió el segmento como la unidad básica de modelado (Definición 2.9). Agregaremos entonces una LEF ($eventList_i$), a cada uno de los segmentos del modelo, esto es, $eventList_i$ contendrá los eventos futuros que se ejecutarán en el segmento s_i.

El evento siguiente a ser ejecutado por el mecanismo de ejecución del módulo *StreetController* es representado por *Ne* tal que, $Ne = \{evt_i | evt_i \in E^2F \wedge \quad evt_i.time = \min\{eventList_i\}_{i=0}^{n}\}$ donde n es el número de segmentos existentes en el sistema ($n = |S|$).

En la Fig. 3.5 se muestra el algoritmo de ejecución para la máquina de simulación con LEF distribuidas y ejecución centralizada. Este algoritmo utiliza una estructura de datos formada por dos listas doblemente enlazadas circulares: lista de eventos potenciales (LEP) que conectan las LEF (o listas de eventos no potenciales) de cada segmento (ver Fig. 3.6). Cada nodo de la estructura representa un evento, a excepción del nodo *rootControl,* que representa el inicio de la LEP, y los nodos *SQS* (*sequencing set*), que representan el inicio de su correspondiente LEF. Los eventos que forman la LEP, están conectados entre sí por los campos *izq.* y *der.* del nodo que los representa. El evento referenciado por el campo *der.* del nodo *rootControl* se encuentra el siguiente evento a ser ejecutado por la máquina de simulación. Los eventos que forman las LEF, están conectados entre sí por los campos *ant.* y *sig.* Estos también son una lista enlazada ordenada que contiene los eventos a ejecutarse en el segmento al que pertenece la LEF. El evento referenciado por el campo *sig.* del nodo *SQS* es el evento de mayor prioridad (menor marca de tiempo) del segmento.

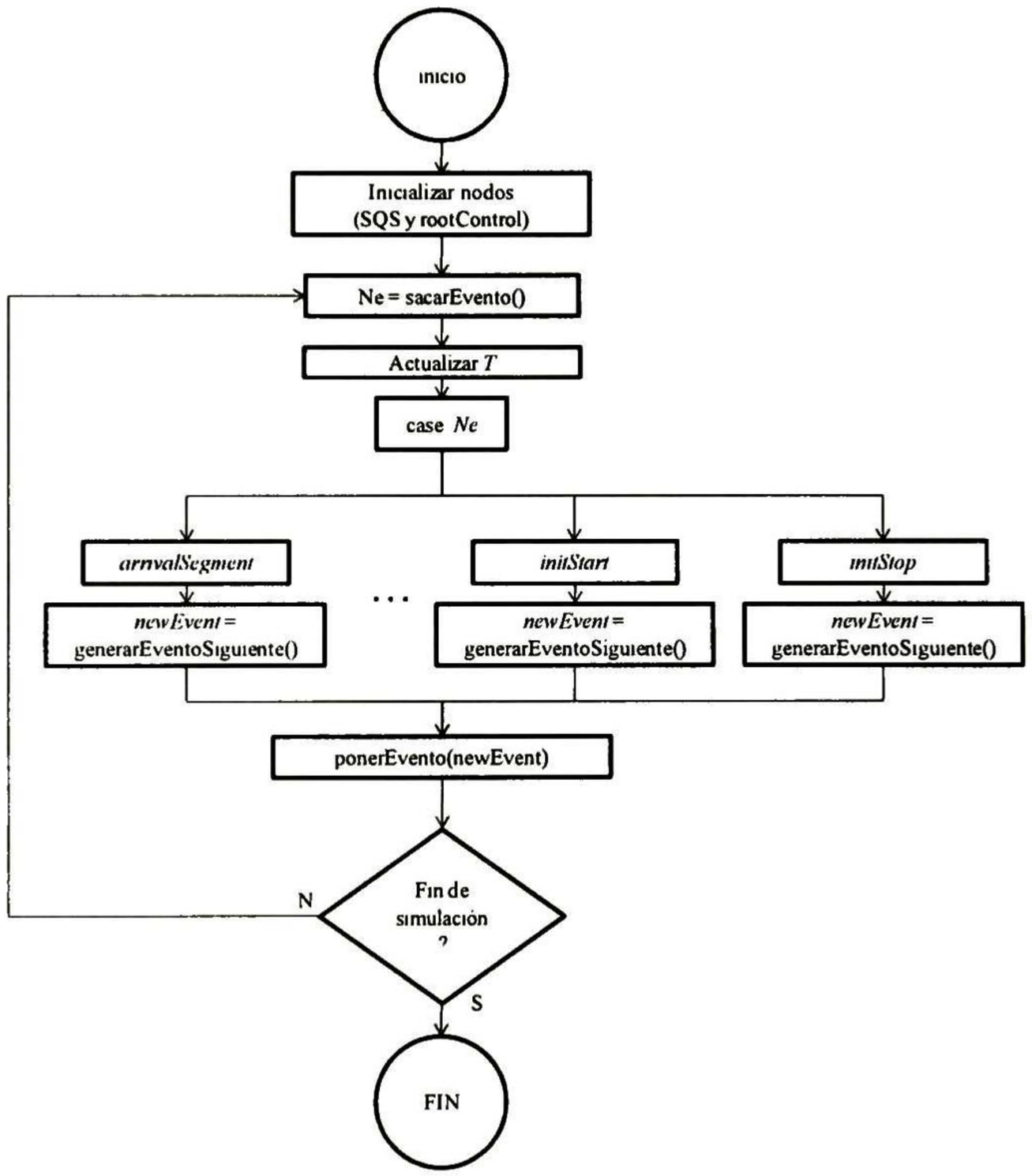


Fig. 3.5: Algoritmo de simulación con LEF distribuidas y ejecución centralizada

A continuación se describen dos de los procedimientos principales del algoritmo utilizado por la máquina de simulación.

Agregar Evento (*Enqueue*).

Al inicio es creado el nodo etiquetado como *rootControl*; este será el nodo que encabeza la lista de eventos potenciales. Inicialmente los campos *izq.* y *der.* de este nodo hacen referencia a sí mismo. También son inicializados los *n* nodos etiquetados como *SQS*, donde $n = |S|$. Inicialmente los campos *sig.* y *ant.*, de este nodo hacen

referencia a sí mismos. Para realizar la inserción del nuevo nodo (evento) en la estructura, se requiere especificar la lista *SQS* en la que será insertado (segmento) y si la marca de tiempo del nuevo evento es la menor en el segmento, entonces será ordenado en forma no decreciente en la LEP y etiquetado como evento potencial del segmento.

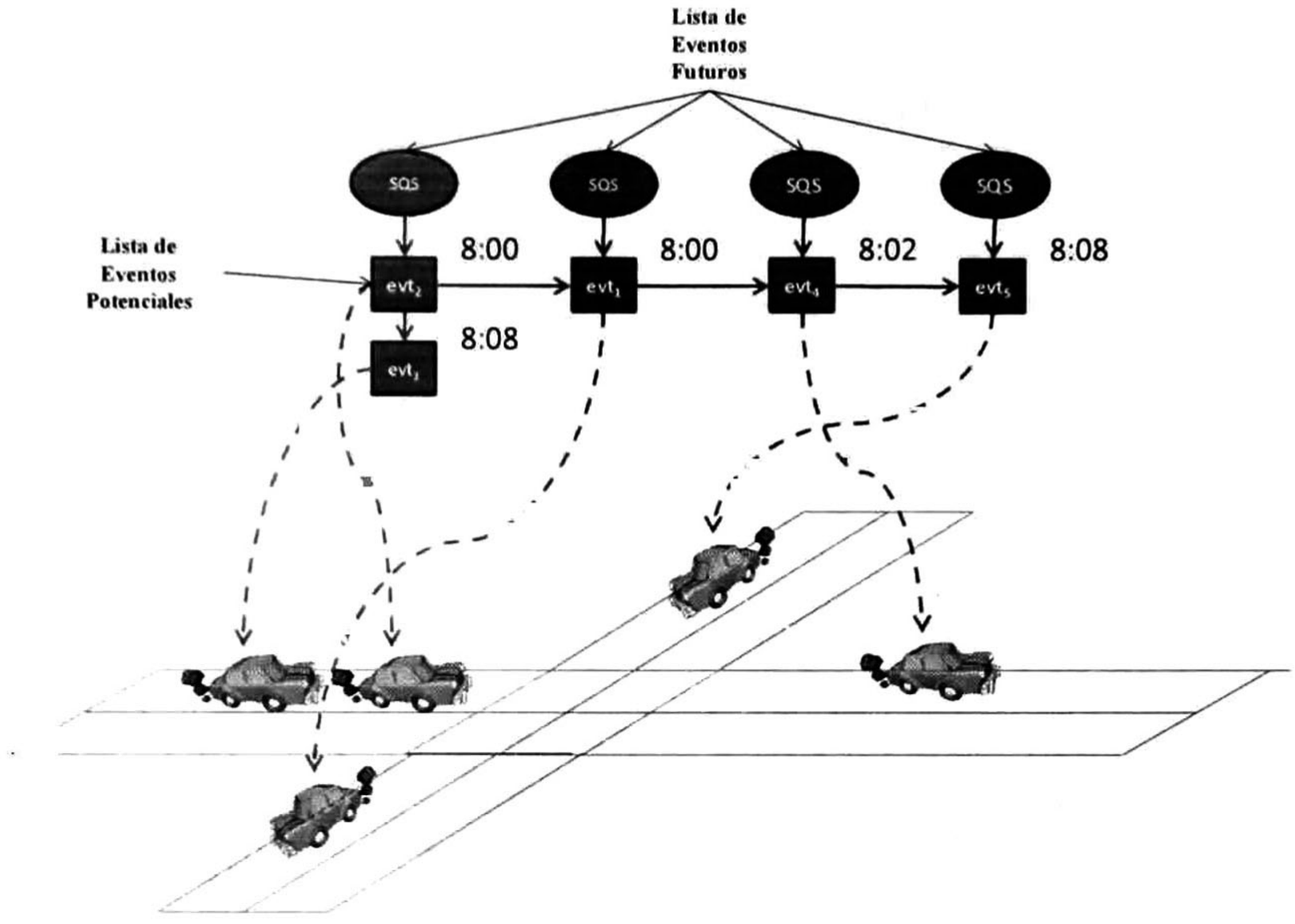


Fig. 3.6: Estructura de *eventos potenciales y futuros de los segmentos*

Ejecutar evento (*Dequeue*)

Para ejecutar algún evento de alguna de la estructura, es necesario sacar el evento potencial que se encuentre referenciado por el campo *der.* del nodo etiquetado como *rootControl*. Si existen más eventos en la lista encabezada por el nodo etiquetado como *SQS*, entonces es necesario determinar el siguiente nodo del segmento con menor marca de tiempo, como evento potencial, y buscar la posición correcta en la lista de eventos potenciales.

Este algoritmo permite la ejecución de eventos de manera secuencial conservando la relación de causalidad válida entre los eventos, a continuación se presentan los mecanismos para la ejecución concurrente y distribuida de los eventos.

3.3.2. Ejecución concurrente

Algunos eventos del modelo del STU pueden ser ejecutados de manera concurrente en la simulación. La concurrencia es una propiedad de los sistemas en los cuales los procesos de cómputo son ejecutados al mismo tiempo y con una alta probabilidad de interactuar entre ellos. Los procesos concurrentes pueden ser ejecutados en el mismo instante de tiempo si se cuenta con procesadores distintos o producir la apariencia de simultaneidad cuando sus pasos de ejecución son intercalados, como es el caso de un sistema multitareas. Ya que los procesos en un sistema concurrente pueden interactuar uno con otro mientras éstos se están ejecutando, el número posible de rutas de ejecución en el sistema es muy alto, resultando en un comportamiento complejo e ineficiente [**105**]. A continuación se presenta una clasificación de los eventos del

modelo del STU que pueden ser ejecutados de manera concurrente: eventos del mismo proceso y eventos de procesos diferentes.

Eventos del mismo proceso

En el modelo del STU presentado en el capítulo anterior, existen eventos que son colocados en la E^2F como resultado de un mismo proceso de razonamiento del agente. En la Fig. 3.7 se muestra la secuencia de eventos el cambio de carril y cambio de segmento (movimientos transversal y longitudinal respectivamente de una entidad). En estos casos, los eventos fueron definidos en un proceso de razonamiento anterior a la ejecución de estos eventos, así que la relación de causalidad fue observada y cuidada por el agente al definir la decisión de cambiar de carril o dejar el segmento. Así pues, estos eventos pueden ser ejecutados al mismo tiempo sin causar problemas de causalidad al sistema. Un evento es propio si pertenece a la misma actividad y proceso de un agente.

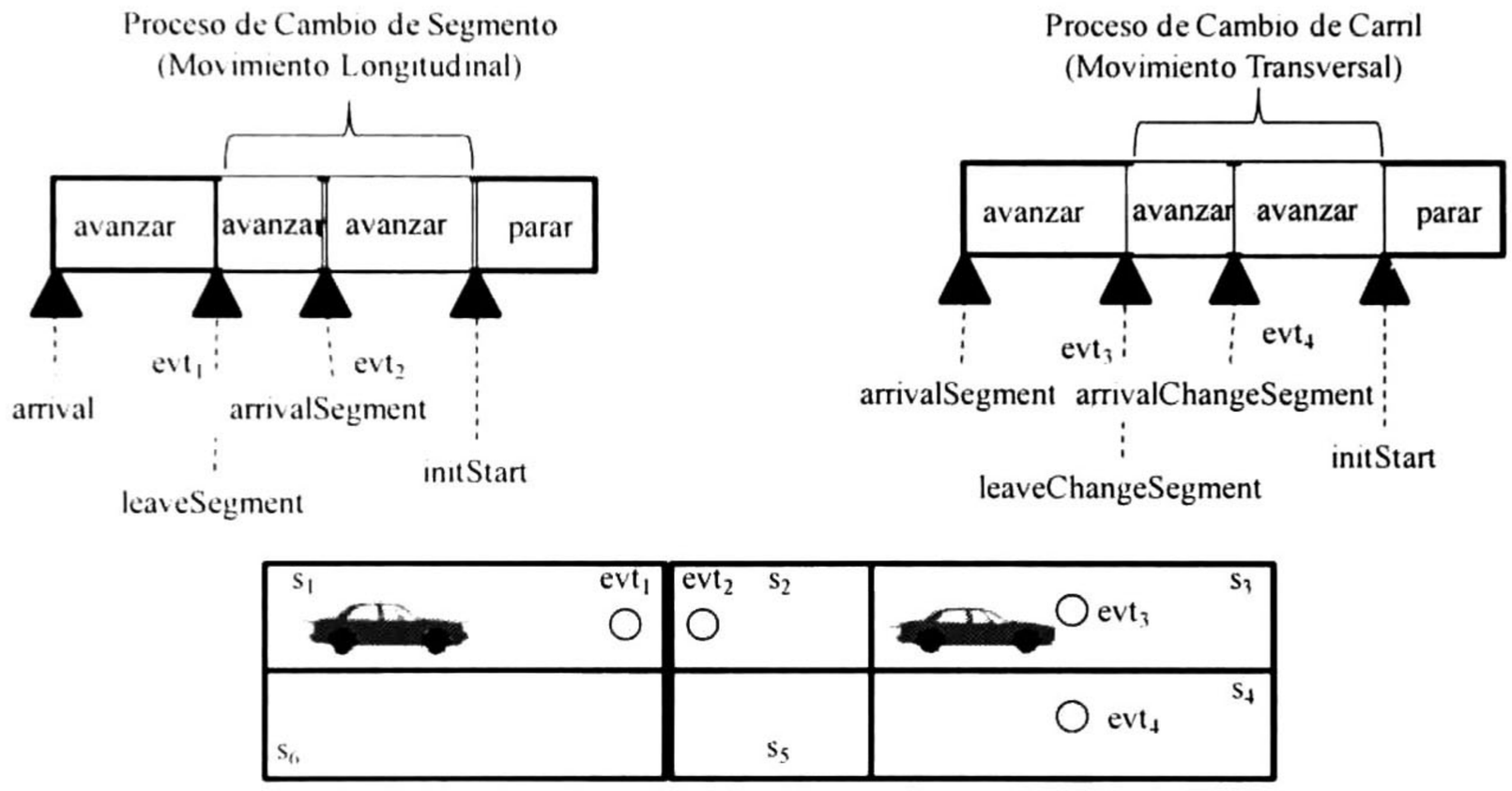


Fig. 3.7: Ejemplo de eventos propios del modelo de STU

Eventos de procesos distintos

Una las tareas más importantes de la máquina de simulación es mantener una relación de causalidad válida entre eventos; esto es, aunque los eventos sean creados dinámicamente, ningún evento con la marca de tiempo $evt_i.time < T$ puede existir en el instante de simulación T. Deben vigilarse las relaciones de causalidad capturadas en la estructura E^2F. Sin embargo, esta restricción no permite la ejecución concurrente de eventos en segmentos distantes y distintos que no generarían problemas de causalidad. Una estrategia posible sería verificar que los recursos (segmento de calle) asignados a los agentes, no generen un problema de *dead-lock* al ejecutar de manera concurrente dos eventos que estarán esperando por el mismo recurso.

La ventaja del modelo de STU propuesto, es que el control de asignación de recursos (espacio disponible en los segmentos) es controlado por el agente durante el proceso de razonamiento, y en caso de existir un conflicto de recurso (intento de usar el mismo espacio al mismo tiempo, o romper una de las LE del sistema), éste es resuelto por el mismo agente. Así pues, lo único por lo que debemos preocuparnos es que la ejecución de un evento no afecte de manera directa la ejecución concurrente de otro

evento. A continuación presentamos la definición de unidad de ejecución que es la base de la estrategia para la ejecución concurrente del modelo.

Definición 3.1. Una unidad de ejecución con respecto al segmento *i* ($UNIT_i$), está formada por aquellos segmentos relacionados con éste a través de las relaciones *NS* y *NC*. Sean los siguientes subconjuntos de *S*, que caracterizan aquellos segmentos relacionados con s_i:

$$SNS = \{s_j | (s_i, s_j) \in NS\}$$
$$CNC = \{s_j | (s_i, s_j) \in NC\}$$
$$NCNS = \{s_k | s_j \in CNC \wedge (s_j, s_k) \in NS\}$$
$$NSNC = \{s_k \mid s_j \in SNS \wedge (s_j, s_k) \in NC\}$$
$$NSNS = \{s_k \mid s_j \in SNS \wedge (s_j, s_k) \in NS\}$$

Entonces,

$$UNIT_i = \{SNS \cup CNC \cup NCNS \cup NSNC \cup NSNS\}.$$

Un segmento solo puede pertenecer a una unidad de ejecución. Este conjunto se actualiza cuando un evento de la E^2F está listo para ejecutarse, y se mantiene hasta que el agente que la inicio termine su proceso de ejecución. Suponiendo que el ejemplo de la Fig. 3.8 tiene las siguientes relaciones $NS(s_{35}) = \{(s_4)(s_{22})\}$, $NC(s_{35}) = \{(s_{36})\}$, $NS(s_{36}) = \{(s_6), (s_{24})\}$, $NS(s_4) = \{(s_3)\}$, $NS(s_6) = \{(s_5)\}$, $NC(s_{22}) = \{(s_{20})\}$, $NS(s_{22}) = \{(s_{23})\}$ entonces:

$$UNIT_{35} = \{(s_3), (s_4), (s_2), (s_5), (s_6), (s_{36}), (s_{24}), (s_{22}), (s_{20}), (s_{23}), (s_{25}))\}$$

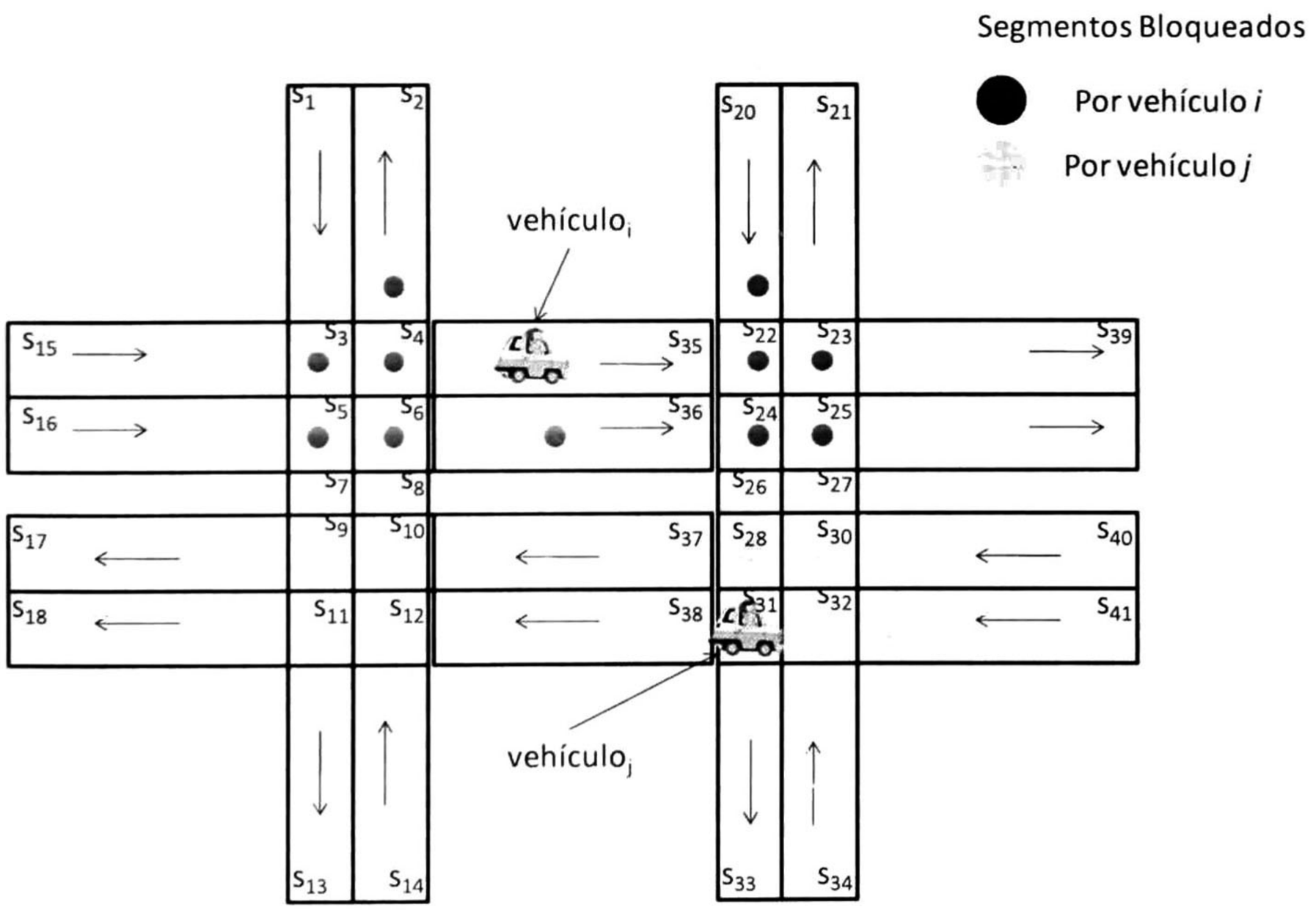


Fig. 3.8: Definición de Unidad.

Usando la Definición 3.1 se puede asegurar que los eventos seleccionados pueden ser ejecutados de manera concurrente y que dicha ejecución no cause un conflicto de causalidad de manera espacial, sin embargo no se asegura una ejecución libre de conflicto causal temporal. Esto es, supongamos que en la simulación tenemos dos segmentos: s_i y s_j. La E^2F del segmento i, $s_i.eventList_i$, contiene un evento evt_i con marca de tiempo $evt_i.time$ = 3:00 am y la estructura de E^2F del segmento j, $s_j.eventList_j$, contiene un evento evt_j cuya marca de tiempo $evt_j.time$ = 10:00 pm. Si ambas E^2F contuvieran solo dichos eventos, sería correcto ejecutarlos de manera concurrente, sin embargo por el algoritmo de ejecución de la máquina de simulación (Fig. 3.5), es posible que $s_j.eventList_j$ contenga más eventos futuros, es decir algún evt_z que cumpla con la siguiente condición: $evt_j.time > evt_z.time > evt_i.time$. Para evitar errores de causalidad temporales, es definido el valor Δ para cada paso de ejecución.

Definición 3.2. Δ es un rango de tiempo mínimo en el que puede ejecutarse eventos sin causar problemas de causalidad temporales. Este valor está definido con respecto al valor mínimo de tiempo requerido por una entidad para trasladarse de un segmento a otro.

Definición 3.3. *History* es el conjunto de eventos cuya marca de tiempo $evt_i.time$ se encuentra en el rango $[T, T+\Delta]$ en un paso de simulación. En la Fig. 3.9 se muestra el conjunto *History* para un paso de simulación.

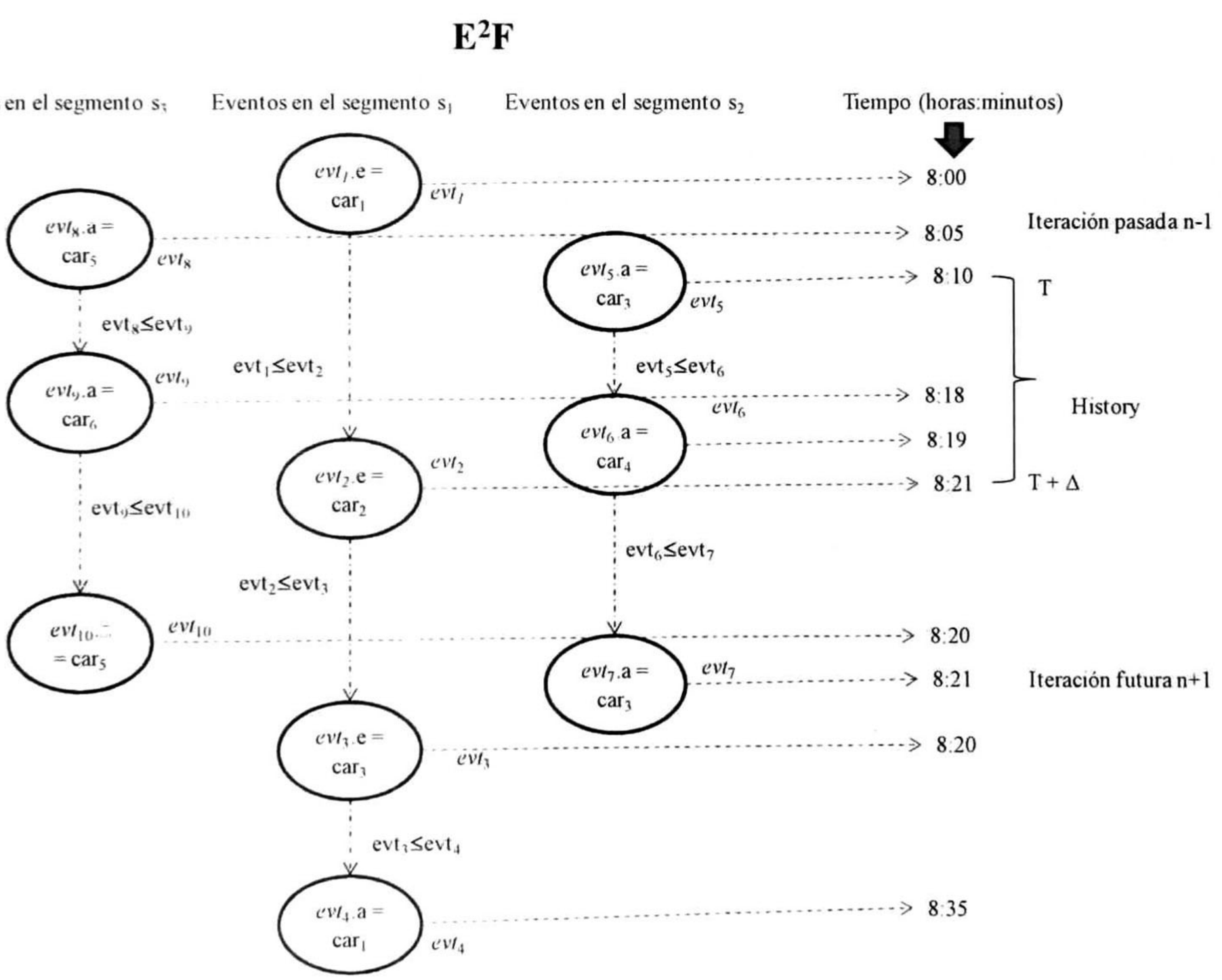


Fig. 3.9: Parte dinámica del sistema

De lo anterior se puede afirmar que dos eventos pueden ejecutarse de manera concurrente si y solo si: pertenecen a diferentes unidades y pertenecen al conjunto *History* del paso de simulación actual o son eventos propios. Es decir, sean evt_i y evt_j

dos eventos a ser ejecutados de manera concurrente en un paso de simulación. La marca de tiempo del evento evt_i es menor que la del evento evt_j ($evt_i.time < evt_j.time$); el evento evt_i pertenece al conjunto $Unit_A$ y el evento evt_j al conjunto $Unit_B$. Por la Definición 3.4, el evento evt_i se encuentra en un segmento distinto al del evento evt_j y por la Definición 3.6, los tiempos de ejecución se encuentran dentro del intervalo $[T, T+\Delta]$ del paso de simulación actual. Entonces evt_i *y* evt_j pueden ejecutarse sin causar problemas de causalidad temporal ni espacial, ya que no existe un evento que sea menor que evt_i *y* evt_j en sus respectivos segmentos.

Usando las definiciones anteriores se define el Algoritmo 3.1. En este algoritmo se describe la ejecución concurrente de la máquina de simulación. Cuando se establece el evento siguiente a ser ejecutado, se bloquean los segmentos que forman un conjunto *Unit* con respecto al segmento.

Algoritmo 3.1 Algoritmo de ejecución concurrente.

```
27: Mientras Existan más eventos Hacer
28:         Establecer T y T+Δ para el paso de simulación actual
29:         Actualizar History
30:         Para cada evt ∈ History generar conjuntos Unit
31:                 Ejecutar de manera concurrente los eventos de cada Unit
32: endPara
33. endMientras
```

3.3.3. Ejecución distribuida

Además del *StreetController* encargado de realizar la ejecución concurrente de eventos del simulador, se agrega un coordinador (*Coordinator*) para cada máquina (host) en la simulación. El coordinador permite coordinar y sincronizar la ejecución de eventos entre diferentes host. Éste tiene su propio hilo de control y puede recibir mensajes de los agentes vehículos cuando su proceso es iniciado. En la Fig. 3.10 se muestra la división de la simulación en dos host y la comunicación del agente vehículo con el coordinador.

Si el agente vehículo desea desplazarse a un segmento que se encuentra en otra sección de red (otro host), entonces la información de estado del segmento destino deberá obtenerla a través del coordinador de host actual. Éste se encargará de mandar un mensaje al coordinador del host destino, de manera que pueda saber si es posible que el agente vehículo pueda desplazarse al segmento destino. De ser posible, el coordinador registra (a través del coordinador destino) el nuevo evento de llegado del agente vehículo a la nueva sección. El protocolo de comunicación entre coordinadores está definido como sigue (ver Fig. 3.11):

1. El agente vehículo verifica si hay espacio disponible en el segmento siguiente que se encuentra en el host destino o receptor.
2. El coordinador envía la solicitud al coordinador receptor de la llegada del vehículo.
3. El coordinador receptor, verificará la existencia de espacio en el segmento destino. De existir espacio generará el nuevo evento para mover el agente vehículo del segmento emisor.

4. En caso de no haber espacio, notificará al coordinador emisor para que sea notificado el agente vehículo y posponga su movimiento.
5. Cada vez que se actualizan los valores de T y $T+\Delta$ el *StreetController*, ejecutor de la en cada host informa a su coordinador para que sea sincronizado con el coordinador vecino.

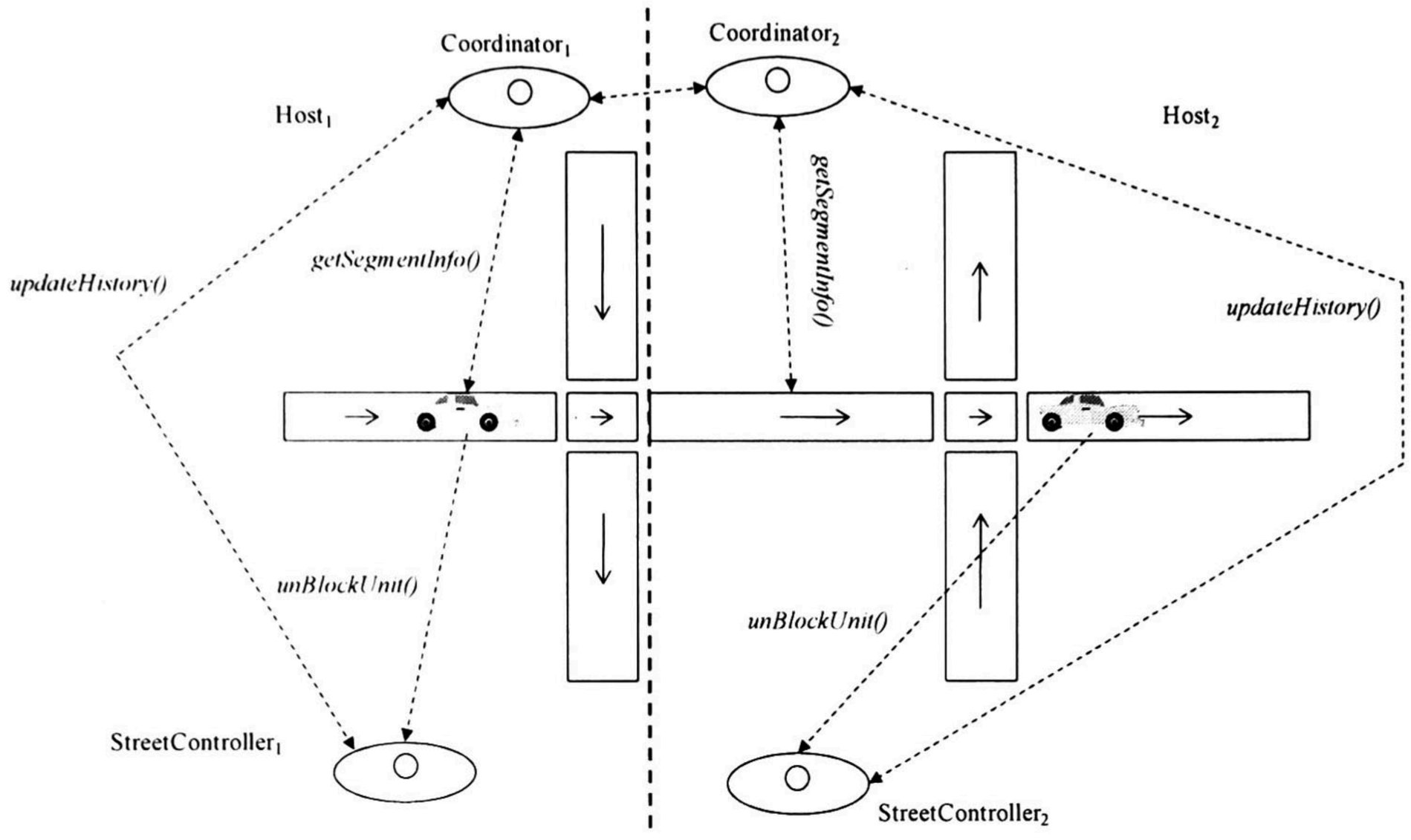


Fig. 3.10: Ejecución distribuida de la simulación de un sistema de tráfico urbano

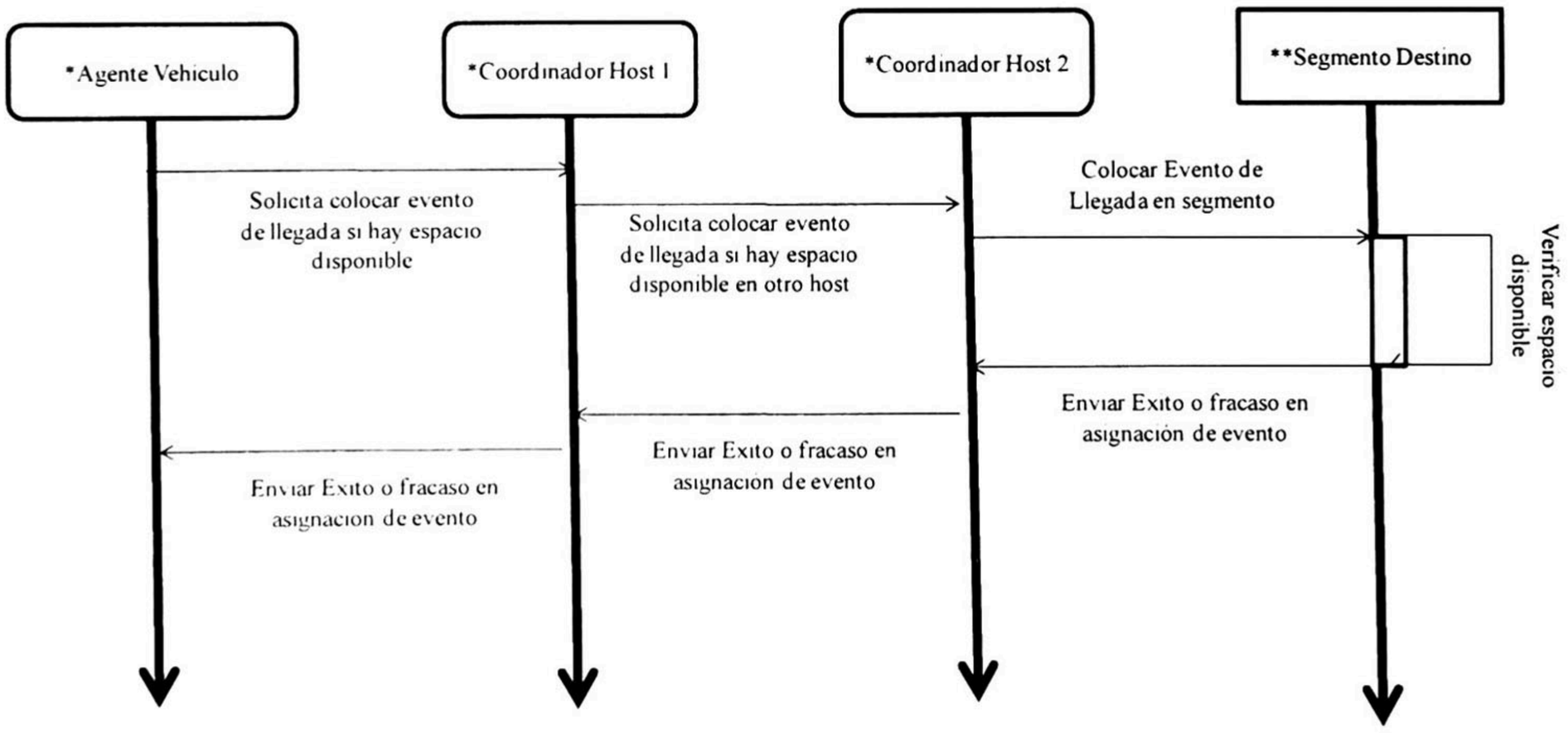


Fig. 3.11: Protocolo de Interacción Distribuida

3.4. Discusión

En este capítulo se presentaron los mecanismos de ejecución concurrente y distribuida para la máquina de simulación del modelo de STU basado en SMA dirigido por eventos. La principal ventaja es la distribución realizada por espacio y por carga de trabajo (eventos a ser ejecutados) en el tiempo. Este enfoque permite que el concepto de PL del área de investigación de la SEDP pueda ser construido de manera dinámica y por lo tanto definidos diferentes mecanismos de ejecución. Entonces la capacidad del modelo de STU para capturar la concurrencia es explotada con los algoritmos propuestos.

CAPITULO 4

IMPLEMENTACIÓN Y RESULTADOS

En este capítulo se presenta la implementación de los modelos de ejecución (máquina de toma de decisiones y máquina de simulación) del modelo enriquecido n-LNS del STU en una biblioteca de clase de Java™. Esta biblioteca implementa la estrategia propuesta en el capítulo anterior para la ejecución concurrente y distribuida del modelo. Además, se presenta un lenguaje para la especificación de las características geográficas, topológicas y de tránsito de un sistema de tráfico urbano (UTYiL) y un caso de estudio para la validación del modelo.

El modelo n-LNS de STU puede ser implementado en alguna plataforma de agentes (e.g. JADE), ya que el modelo no es dependiente de ningún lenguaje en particular. Con el fin de tener control total sobre la plataforma usada e implementar la estrategia de ejecución concurrente y distribuida propuesta en este trabajo de tesis, se desarrolla una plataforma propia usando el lenguaje Java™. A continuación se presenta el diseño e implementación de dicha plataforma.

4.1. Biblioteca de Simulación

La biblioteca implementa los componentes del modelo n-LNS usando Java™. La biblioteca incluye una interfaz XML para la descripción de los modelos usando el lenguaje UTYiL para el modelado de datos, el cual será descrito posteriormente.

La arquitectura de la biblioteca es mostrada en la Fig. 4.1. Consiste de tres módulos: el generador de vehículos, el control de semáforos y la máquina de simulación. La máquina de simulación genera las estructuras necesarias que representan el modelo STU de acuerdo a lo descrito por UTYiL. El módulo de control de tráfico usa el modelo del STU descrito con UTYiL para conocer la estructura de la red vial y de acuerdo a la estrategia de control asignada, colocar los eventos en los segmentos de la red vial. La descripción del STU usando UTYiL también es usada por el módulo generador de vehículos para conocer en qué posición de los segmentos se colocarán las nuevas entidades (vehículos).

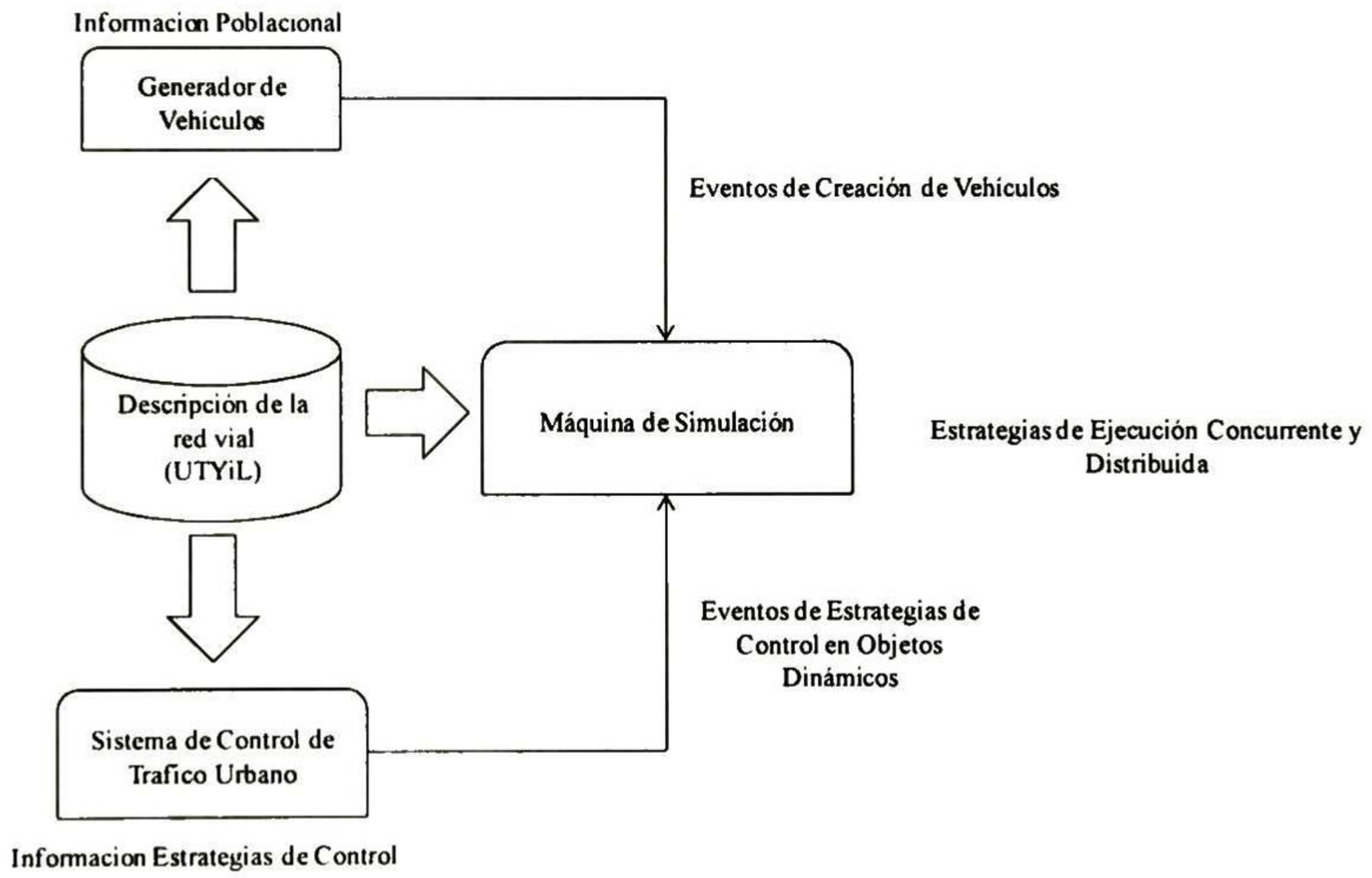


Fig. 4.1: Arquitectura del Simulador de STU

La biblioteca se basa en principios de diseño de la programación orientada a objetos (POO). La biblioteca pretende satisfacer los siguientes principios de diseño:

- *Uso de multi-hilos.* El principal objetivo del simulador es aprovechar la capacidad concurrente de la computadora.
- *Ejecución Paralelo.* Es deseable que la biblioteca pueda ser usada en ambientes distribuidos aplicando mecanismos de la descomposición de dominios. Este nivel de paralelismo puede incrementar el desempeño y la cantidad de memoria empleada, así como la capacidad de procesamiento para el modelado de STU.

A continuación se presenta el diseño de la biblioteca del simulador de STU. Para documentar el diseño del simulador se toma como base el estándar de diseño OMT, ya que éste entrega una clara referencia a la utilización de modelos de objeto y modelos dinámicos.

4.2. Modelo de Objetos

Para el modelo de objetos se usan los diagramas de clases, junto con un diccionario de datos para formalizar el significado de las clases sus atributos y métodos.

4.2.1. El ambiente

La red de ambiente *EnvironmentNET1* se implementa como una clase contenedora de instancias de la clase *Segment*, que representa a los lugares de la red. Las conexiones entre las redes son capturadas como vectores en cada instancia de la clase *Segment* (ver Fig. 4.2).

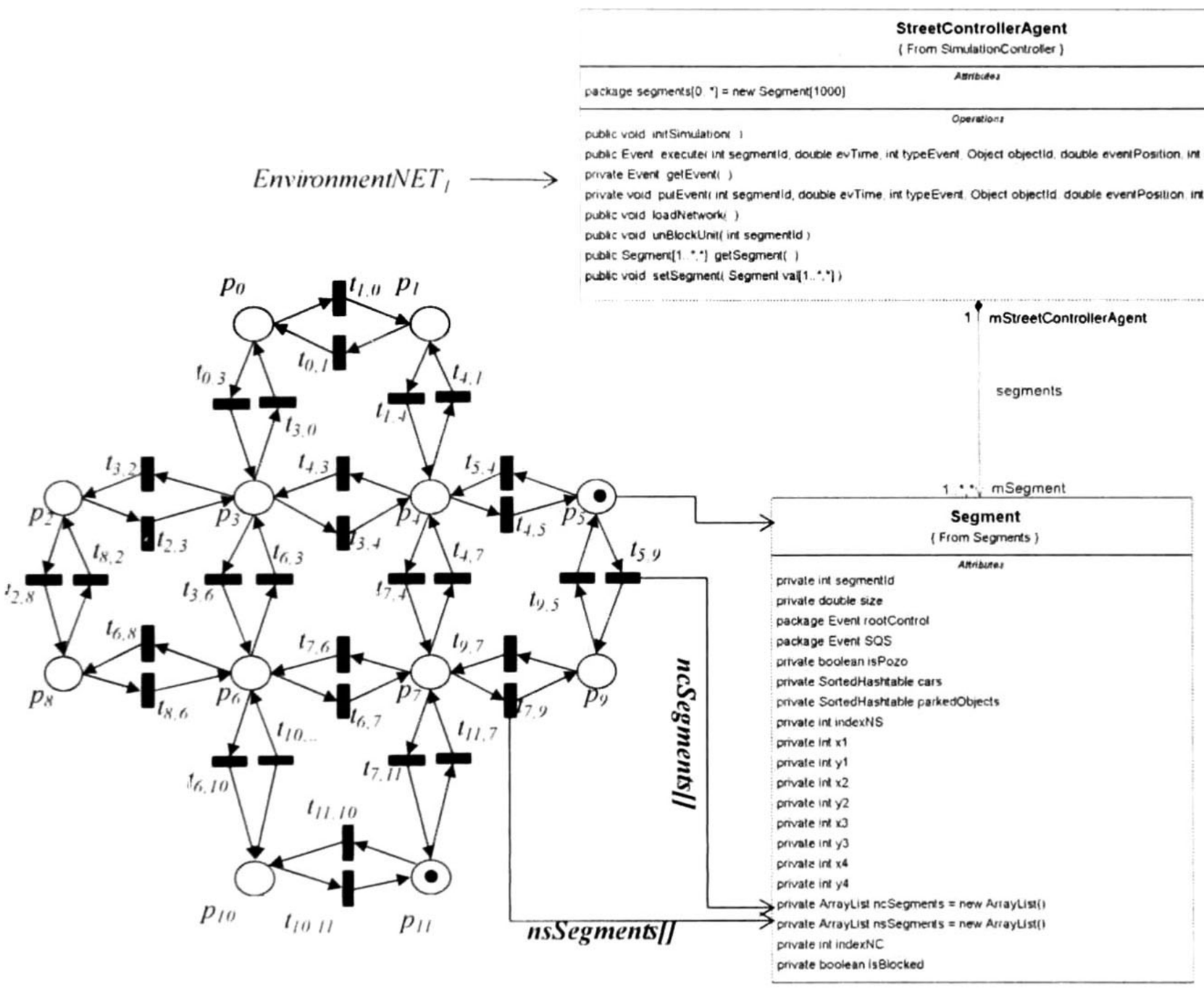


Fig. 4.2: Implementación de la red ambiente

4.2.2. Entidades

La red de Petri que describe los atributos, parámetros y proceso que se encuentra ejecutando la entidad, se implementa como una clase *Vehicle*. Las redes que describen sus habilidades y actividades como *threads*, que son ejecutados solo cuando la entidad requiere ejecutar la actividad y realizar un proceso de toma de decisión.

ProcessNet$_3$ (estado del vehículo)

Esta red que describe características, base de conocimiento, atributos y los eventos que puede ejecutar, es descrita como la clase llamada *Vehicle* (ver Fig. 4.3).

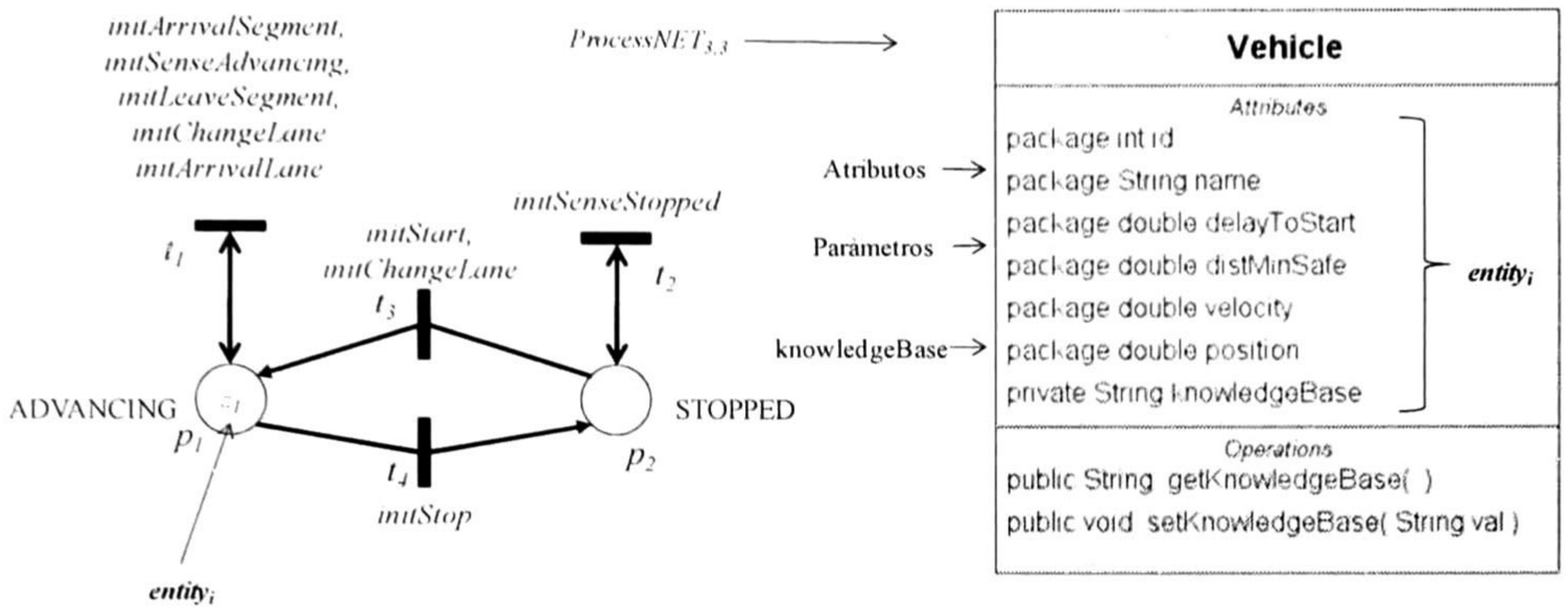


Fig. 4.3: Implementación de la red *ProcessNET*$_3$.

Redes ActivityNET$_3$ y SkillNET$_3$

En el capítulo anterior se asociaron las transiciones que sincronizan las actividades y el proceso de una entidad, con los eventos de un grafo llamado E^2F. Esta sincronización ejecuta la actividad (*ActivityNet*$_3$) y usa alguna red de habilidad (*SkillNET*$_3$) y modifica el proceso de la entidad (*ProcessNET*$_3$). Estos eventos (sincronizaciones) son implementados con la clase llamada como la actividad que sincroniza las habilidades y el proceso de la entidad (e.g. *StartActivityNet*$_{3,1}$ --> clase *carStart*). Todas estas clases heredan de la clase *Event* (ver Fig. 4.4)

Por lo anterior, las redes *ActivityNET*$_3$ son implementadas como clases que heredan de la clase *Event*. Además, se les asocia a la implementación las redes *SkillNET*$_3$, de manera que las clases que implementen la clase *Event*, estarán implementando el comportamiento de las entidades. La ejecución de cada uno de estos eventos estará relacionada con la lista de entidades (vehículos) que están contenidos en cada segmento en las listas de tipo *SortedHashTable*. Por ejemplo la actividad *StartActivtyNET*$_{3,1}$, tiene su clase *CarStart*. Cada instancia de estas clases es almacenada en la estructura SQS de la clase *Segment*. De esta manera cuando es ejecutado el evento, la máquina de simulación le asigna un hilo de ejecución a la actividad.

4.2.3. Implementación mecanismo de toma de decisión

Las transiciones de las redes *SkillNET*$_3$ que requieran aplicar un proceso de toma de decisión son asociadas a un proceso que se encarga de retornar el valor de las variables necesarias para que la red pueda tomar una decisión. En la implementación las redes *ActivityNet*$_3$ y las redes *SkillNET*$_3$ son implementadas como la clase *Event*. Los algoritmos 4.3 y 4.4 describen la implementación del proceso ejecutado cuando alguna instancia de la clase *Event* requiere tomar una decisión.

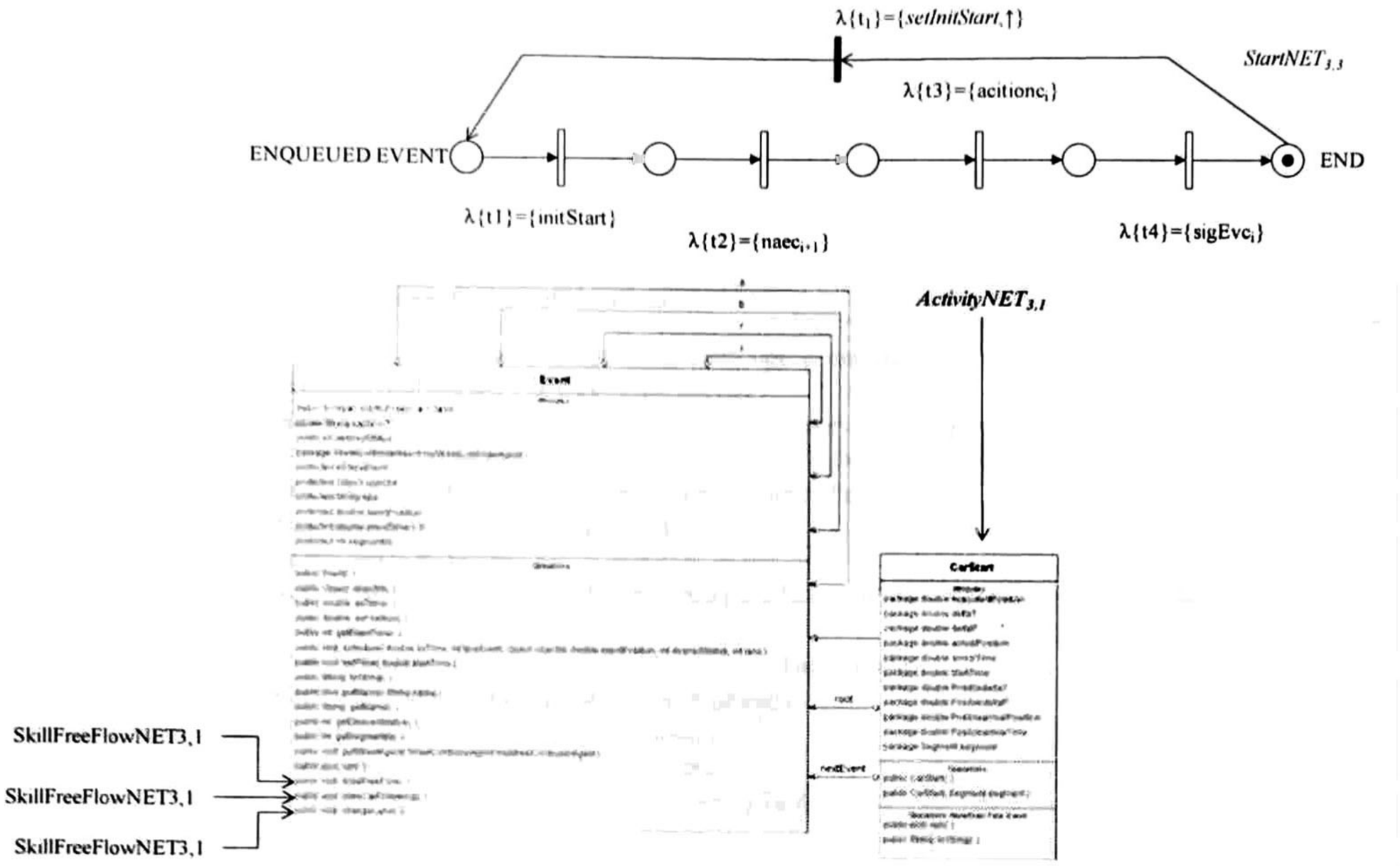


Fig. 4.4: Implementación de los eventos de la estructura E^2F.

Algoritmo 4.3 Mecanismo de Toma de Decisión

Func: carArrival():Thread
{Agregar un nuevo carro al segmento actual}
actualEvent.segment.putCar((Car) objectId(), new Double(0.0))
set evaluatedPosition = segment.getSize()
{aquí es la implementación de reglas para el primer caso o interaccion}
{Calcular en tiempo de arrive para siguiente evento posicion}
set arrivalPosition = nextEvent.evPosition() – car.getDisMinSafe()
set deltaP = arrivalPosition – actualEvent.evPosition();
set deltaT = (deltaP * 3600) / car.getVelocity();
set arrivalTime = deltaT + actualEvent.evTime();
{Buscar eventos osibles en rango}
while()do
case: vehicle-vehile
{implementación reglas}
case: vehicle-light
{implementación reglas}
case: vehicle-otra entidad
{implementar reglas obtenidas usando metodología}
case: no_events
set evaluatedPosition = nextEvent.evPosition()
end while

Algoritmo 4.4 Implementacion Reglas vehicle-vehicle

{implementación de la regla R_1}
if(arrivalTime >= nextEvent.evTime() AND nextEvent = BE;CE;WSE) **then**
set newEvent = WSE
set startTime = arrivalTime
end if
{Implementacion de la regla R_2}
if(arrivalTime< nextEvent.evTime() AND nextEvent = BE)**then**
set startTime = nextEvent.evTime() + car.getDelaytoStart()
set newEvent = SE

```
endif
{Implementacion de la regla R4}
if(arrivalTime< nextEvent.evTime() AND nextEvent = CE;WSE)then
set startTime = nextEvent.evTime()
set newEvent = WSE
endif
```

Algoritmo 4.5 Implementacion Reglas vehicle-semáforo

```
{Si el evento es de un semáforo, aquí es para otro tipo de entidad}
set posibleArrivalPosition = nextEvent.evPosition()
set posibleDeltaP = posibleArrivalPosition – actualEvent.evPosition()
set posibleDeltaT = (posibleDeltaP * 3600) / car.getVelocity()
set posibleArrivalTime = posibleDeltaT + actualEvent.evTime()
{analizar si ya habia otro evento definido}
{Implementación de la regla R4}
if(nextEvent.getDesiredStatus = VERDE AND posibleArrivalTime <nextEvent.evTime())then
set newEvent = SE
set arrivalPosition =  posibleArrivalPosition – car.getDisMinSafe()
set deltaP = arrivalPosition – actualEvent.evPosition()
set deltaT = (deltaP * 3600) / car.getVelocity()
set arrivalTime = deltaT + actualEvent.evTime()
set startTime = nextEvent.evTime() + car.getDelayToStart()
endif
```

4.2.4. Relación causal E^2F

El segmento es representado con la clase *Segment* (ver Fig. 4.5), cada una tiene una estructura llamada SQS, que permite la implementación del $eventList_i$ del segmento s_i. La lista de entidades en el segmento es implementada dentro de la estructura *Segment*, como una lista de objetos *cars* del tipo *SortedHashTable*. A continuación se describe la implementación de la estructura E^2F utilizando listas doblemente enlazadas.

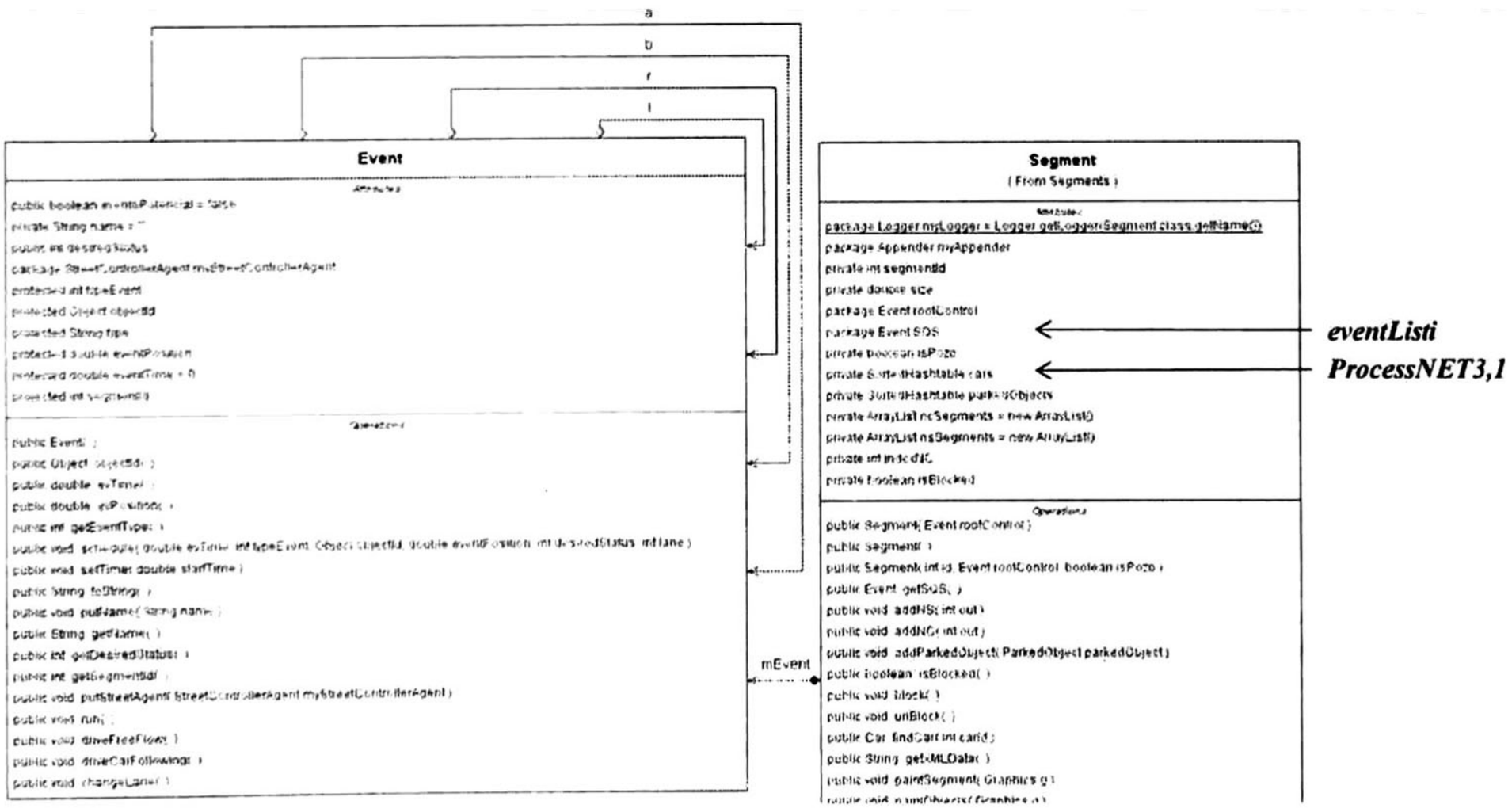


Fig. 4.5: Implementación de la estructura E^2F.

Lista enlazada dividida por espacio y tiempo

Podemos caracterizar dos listas doblemente enlazadas circulares de la estructura de eventos futuros, el de eventos potenciales y el de eventos no potenciales (ver Fig. 4.6):

1. Los nodos conectados entre sí por los campos izquierda y derecha, representan el hilo principal, en el cual los eventos con los tiempos mínimos son ordenados y almacenados. Esta lista principal es una lista enlazada ordenada, cuyo nodo inicial está etiquetado como *rootControl*, a la derecha de este nodo se encuentra el evento siguiente menor de la lista y a la izquierda el mayor de todos.
2. Los eventos no potenciales están formados por todos aquellos nodos conectados entre sí por los campos anterior y siguiente. Estos también son una lista enlazada ordenada que contiene los eventos con respecto a "espacios con comportamiento secuencial" y cuyo nodo principal está etiquetado como *SQS* (por *sequencing set*) y en el campo siguiente encontraremos el nodo con el mayor tiempo de esta lista y con nodo anterior el nodo con menor tiempo de esta lista.

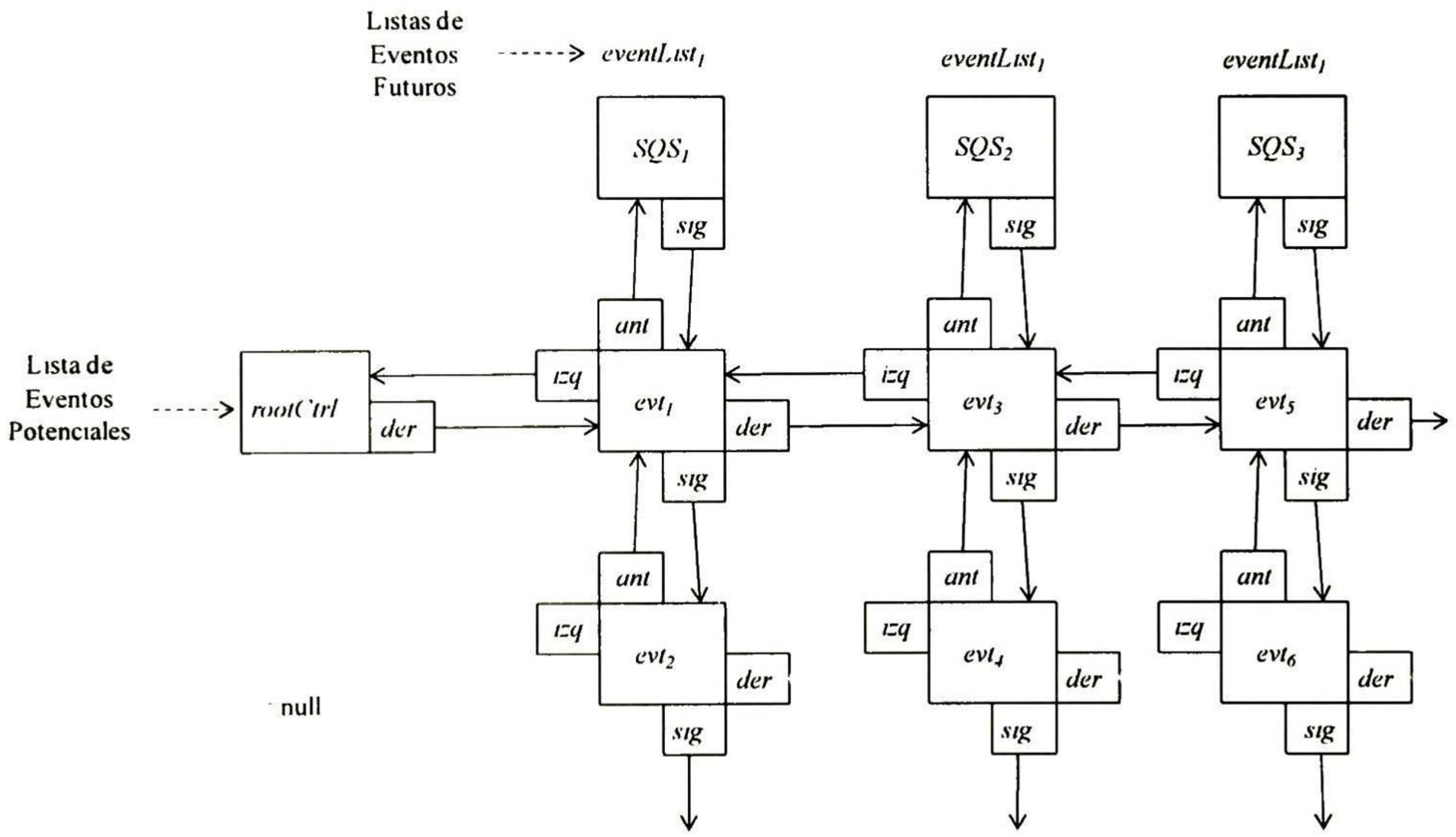


Fig. 4.6: Estructuras que implementan la E²F.

4.2.5. Implementación de la arquitectura distribuida del simulador

En la Fig. 4.7 se muestra la arquitectura distribuida del simulador. Los módulos de generación de vehículo, control de tráfico, y máquina de simulación son distribuidos en computadoras conectadas en red. Para realizar la distribución de la simulación se utiliza un software de conectividad nombrado *middleware*. El *middleware* permite la comunicación transparente entre diferentes computadoras distribuidas a lo largo de la red (Internet, Intranet, etc.). La aplicación distribuye las tareas de cómputo a cualquier

nodo de la red (computadora) que tenga la capacidad para ejecutarla; entonces retornar los resultados a la aplicación.

De varias alternativas de herramientas de comunicación, tales como MPI, PVM, *Corba*, *ProActive*, hemos seleccionado *ProActive* que es un middleware open-source que cumple con estos requerimientos. *Proactive* es un conjunto de conceptos de programación, metodologías y un marco de desarrollo para la construcción de aplicaciones meta-computacionales, que se adapta a la naturaleza jerárquica y heterogénea de las aplicaciones distribuidas. Está basado en Java™ (modelo de programación y ambiente de desarrollo) [**106**].

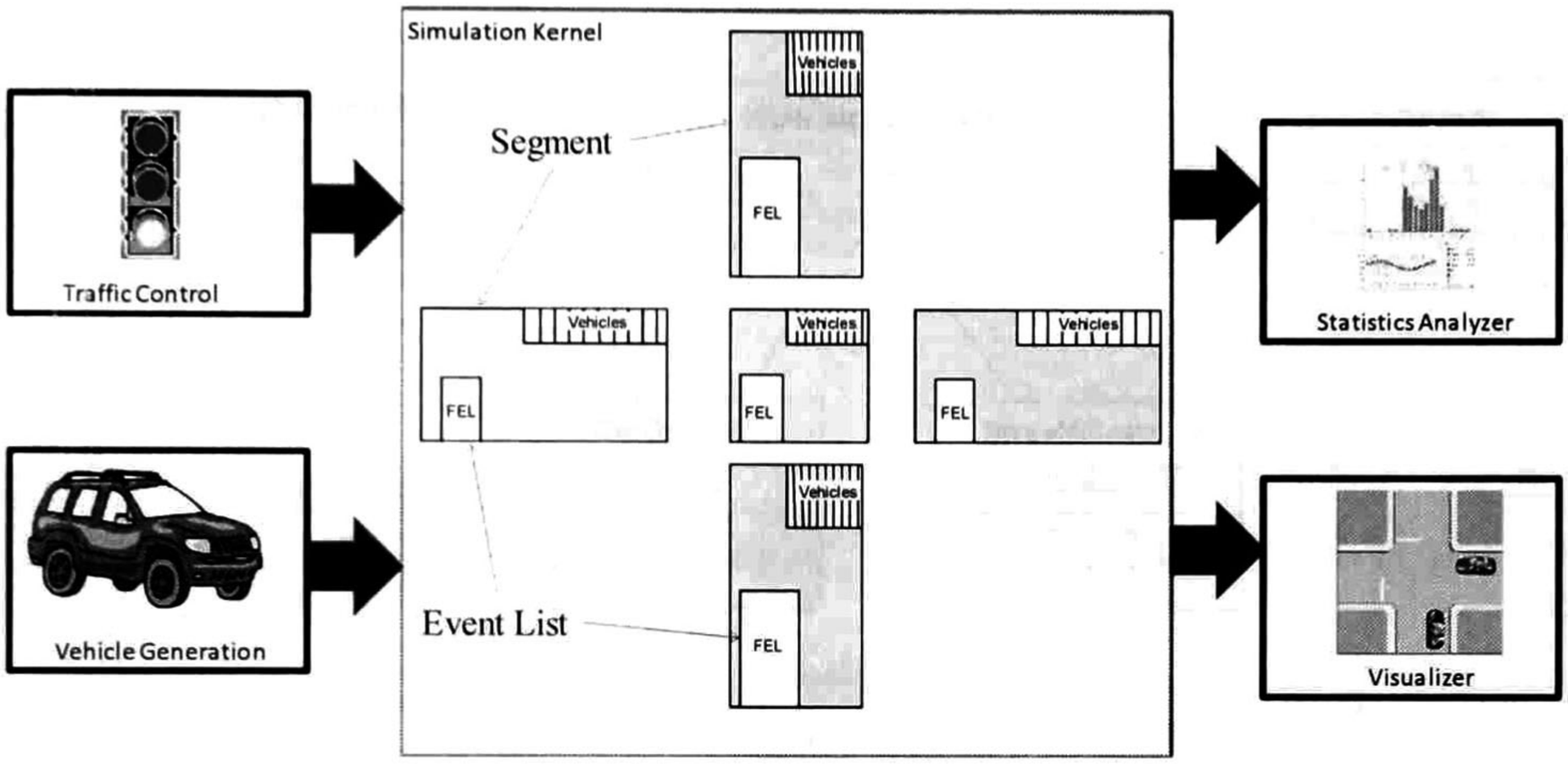


Fig. 4.7: Arquitectura Distribuida del Simulador

Cada módulo es un objeto activo, los cuales son las unidades básicas para actividades y distribución usadas para la construcción de aplicaciones concurrentes usando *ProActive*. Un objeto activo tiene su propio hilo de ejecución. Este hilo solo ejecuta los métodos invocados sobre el objetivo activo por otros objetos activos y aquellos objetos pasivo del subsistema que pertenezcan al objeto activo [**106**]. Cada módulo puede atender requisiciones externas. El generador de vehículos y el control de tráfico están conectados a la máquina de simulación, la cual atiende los eventos de cada uno de estos módulos.

En la Fig. 4.8, se muestra el diagrama de clases del simulador. Las clases *ligthControlAgent*, *AnalyzerAgent*, *SimulatorControler*, *StreetControllerAgent* y el *carGeneratorAgen*t implementan la interfaz *RunActive* de la biblioteca *ProActive*, con esto se permite que cada módulo tenga un hilo individual, una cola de mensajes externos y la habilidad de migrar (movilidad).

El módulo *SimulatorController* lleva el control de creación de cada uno de los módulos del simulador.

El módulo *carGenerator* puede utilizar una base de datos de información demográfica o implementar una distribución de Poisson para generar los vehículos durante el periodo de simulación. Este modulo envía los eventos de creación de vehículos a través de mensajes al *StreetController*.

El módulo *ligthControl* permite generar eventos para todos los semáforos en los segmentos y retroalimentarse de la información generada por cada vehículo en el segmento.

El módulo *analyzer* permite obtener estadísticas a partir de la información generada por el simulador, tales como tiempos de viaje, flujo, densidad, etc.

La máquina de simulación es implementada con el módulo *streetController*. Este modulo se encarga de la creación de los segmentos del simulador y de la implementación de la máquina de simulación.

La clase *StreetController* es la encargada de ejecutar la E2F, siguiendo las reglas de causalidad impuestas. En esta clase el método *run()*, puede ser modificado para probar diferentes estrategias de ejecución. La ejecución básica es la secuencial, se tienen dos métodos el *executeEvent()* controlado por la máquina de simulación que permite sacar el siguiente evento a ser ejecutado, y el método *putEvent()* que es usado por las entidades cuando desean colocar una nueva acción (evento) a ser ejecutado.

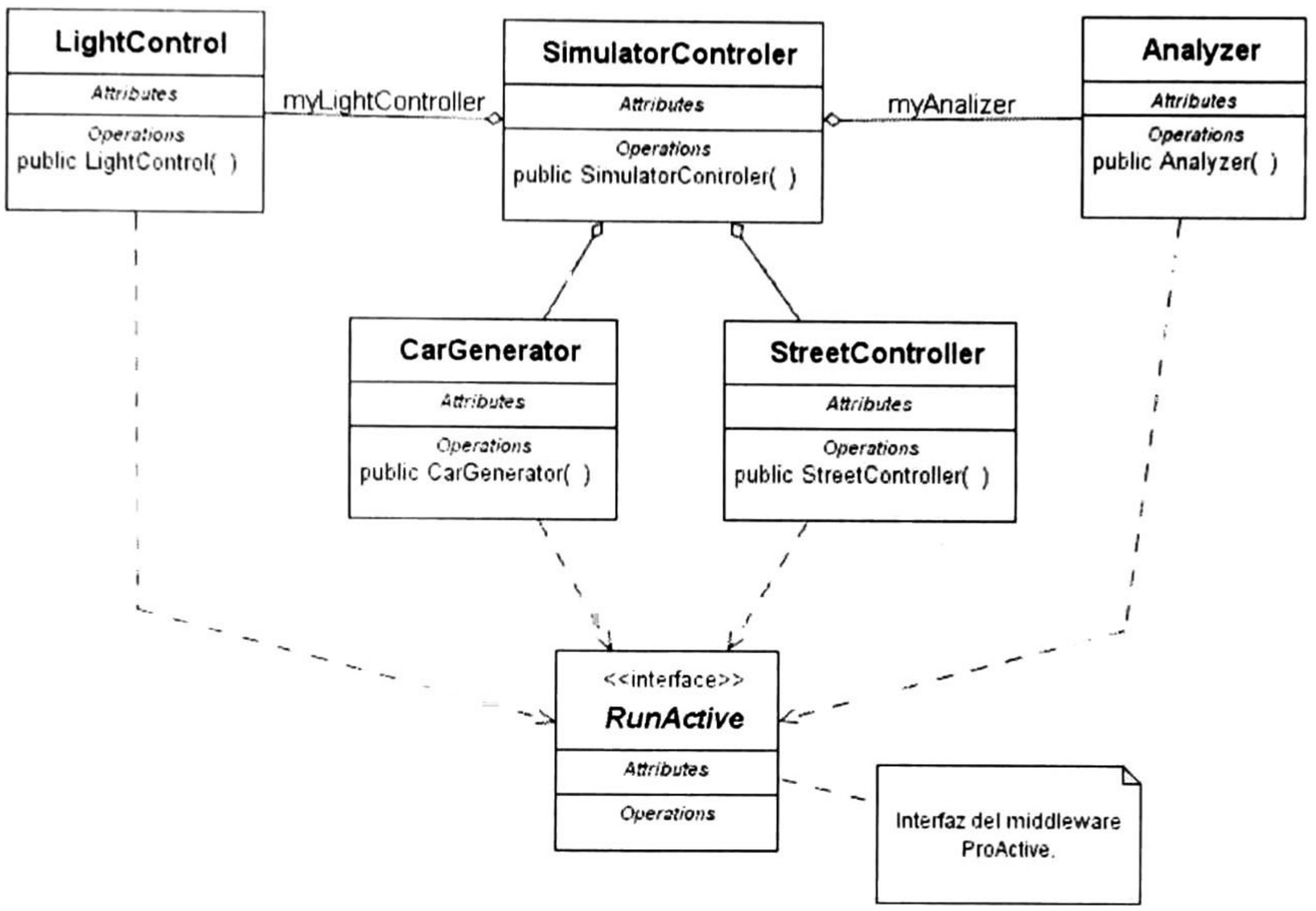


Fig. 4.8: Diagrama de clases del simulador de STU

4.2.6. Diccionario de datos

En la tabla 4.1 se describe el diccionario de datos del simulador.

Clase	Atributos	Operaciones
SimulatorController: Se encarga de generar los módulos de creación de vehículos, de generación de eventos de cambio de luz en los semáforos y de iniciar la máquina de simulación (*streetController*)	***ControllerList***: Contiene la lista de módulos que son controlados durante la simulación.	***createStreetController():*** Crea la máquina de simulación. ***createCarGenerator():*** Crea el módulo para la creación de vehículos. ***createLightControl()***: Crea el módulo para la aplicación de estrategias de control. ***runActivity():*** Método implementado para la ejecución del controlador, de la interfaz del middleware ProActive.
StreetController: Lleva el control de ejecución de los	***rootControl***: Evento que inicia la lista de eventos potenciales.	***initSimulation***(): Inicia la máquina de simulación a sacar

eventos durante la simulación implementando estrategias de ejecución concurrente y distribuida. Recibe los eventos generados por cada entidad durante la simulación.	***segments***: Arreglo que contiene la referencia a los objetos tipo *Segment*. ***isActive***: variable que permite que la máquina de simulación continue ejecutandose.	eventos de la lista de eventos potenciales ***loadNetwork***(): Interpreta el archivo *network.xls*, y genera los segmentos con sus conexiones correspondientes. ***putEvent():*** Permite que los eventos generados por las entidades durante su proceso de toma de decisión sean agregados a la lista de eventos futuros y/o lista de eventos potenciales ***executeEvent***(): Genera un *thread* para la ejecución del evento siguiente a ser ejectuado. ***setActive():*** Activa la simulación.
LightControl: Genera los eventos de cambio de luz en los semáforos de acuerdo a la estrategia de control establecida.	***simCtrl***: Referencia a la instancia de la clase *SimulatorController*. ***strCtrl:*** Referencia a la máquina de simulación *StreetController*.	***LightControl():*** Constructor de la clase. ***LoadNetwork():*** Carga la red descrita en el archivo *network.xml* en el lenguaje UTYiL. ***runActivity():*** Método que inicia el proceso de generación de eventos de los semáforos para ser colocados en el *StreetController*.
CarGenerator: Genera los eventos de generación de nuevos vehículos en la simulación de acuerdo a la distribución pre-establecida.	***simCtrl***: Referencia a la instancia de la clase *SimulatorController*. ***strCtrl:*** Referencia a la máquina de simulación *StreetController*.	***CarGenerator():*** Constructor de la clase. ***LoadNetwork():*** Carga la red descrita en el archivo *network.xml* en el lenguaje UTYiL. ***runActivity():*** Método que inicia el proceso de generación de eventos de creación de vehículos para ser colocados en el *StreetController*.
Analyzer: Analiza el archivo de salida (log.dat) de la simulación y obtiene los valores de flujo y densidad para los segmentos definidos.	***simCtrl***: Referencia a la instancia de la clase *SimulatorController*. ***strCtrl:*** Referencia a la máquina de simulación *StreetController*.	***Analyzer():*** Constructor de la clase. ***LoadNetwork():*** Carga la red descrita en el archivo *network.xml* en el lenguaje UTYiL. ***runActivity():*** Método que inicia el proceso evaluación de resultados contenidos en el archivo log.dat.
Segment: Describe la unidad de modelado básica del modelo n-LNS. Contiene la lista de eventos futuros del segmento (*eventList*$_i$) y la lista de vehículos del segmento (*ProcessNET*$_3$).	***SQS:*** Variable tipo *Event*, que establece el inicio de la lista de eventos futuros a ejecutarse en el segmento. ***cars:*** Variable tipo *SortedHashTable*, que contiene los objetos tipo *Vehicle* de las entidades contenidas en el segmento. ***parkedObjects:*** Arreglo que contiene a los objetos del	***Segment():*** Constructor de la clase. Inicializa el *id* del segmento, establece la relación con el *rootControl*, e inicializa la variable *isPozo*. ***addNC():*** Agrega una conexión tipo NC al segmento. ***addNS():*** Agrega una conexión NS al segmento. ***blockSegment***(): Establece la variable *isBlocked* a *True*.

	segmento. ***rootControl***: Referencia al evento *rootControl* de la lista de eventos potenciales. ***ncSegments:*** Arreglo que contiene las referencias a los segmentos conectados de forma NC con el segmento. ***nsSegments***: Arreglo que contiene las referencias a los segmentos conectados de forma NS con el segmento.	***unBlockSegment():*** Estable la variable *isBlocked* a *False*.
Vehicle: Describe los parámetros y atributos del vehículo.	***position***: Posición del vehículo con respecto al segmento en el que se encuentra. ***name***: Identificador único del vehículo. ***velocity***: Velocidad actual del vehículo. ***delayToStart***: Factor percepción-reacción del conductor del vehículo. ***distMinSafe***: Distancia de seguridad mínima entre vehículos.	*getters* y *setters* para los atributos.
CarStart: Describe la actividad de arranque del vehículo, las habilidades del vehículo y su proceso de toma de decisión.	***arrivalPosition***: Posición en la que iniciará el arranque del vehículo ***arrivalTime***: Tiempo de arranque del vehículo ***deltaP***: Diferencia de la posición anterior al arranque (siempre será cero, ya que siempre estará en reposo total) ***deltaT***: Diferencia de tiempo con respecto al evento inmediato anterior. ***nextEvent***: Siguiente evento a ejecutar resultado del proceso de toma de decisión, después de la ejecución de este evento ***root***: Referencia al evento *rootControl*. ***segment***: Referencia al segmento en el que ocurrirá el evento.	***carstart()***: Constructor del evento. Inicializa el segmento en el que ocurrirá el evento, y el vehículo que lo generó. ***run()***: Inicia la secuencia de tareas de la actividad y el proceso de toma de decisión de la entidad.

4.1: Diccionario de datos del simulador

4.3. Modelo dinámico

En esta sección se presentan los modelos dinámicos de la biblioteca de simulación. Los diagramas de secuencia nos permiten describir el proceso de interacción que tiene cada una de las instancias existentes durante la simulación, además los algoritmos de ejecución de cada proceso son descritos.

4.3.1. Diagrama de secuencia

Las figuras Fig. 4.9 y Fig. 4.10 muestra los diagramas de secuencia para la inicialización de la simulación y la ejecución de eventos en la estructura de eventos futuros E^2F.

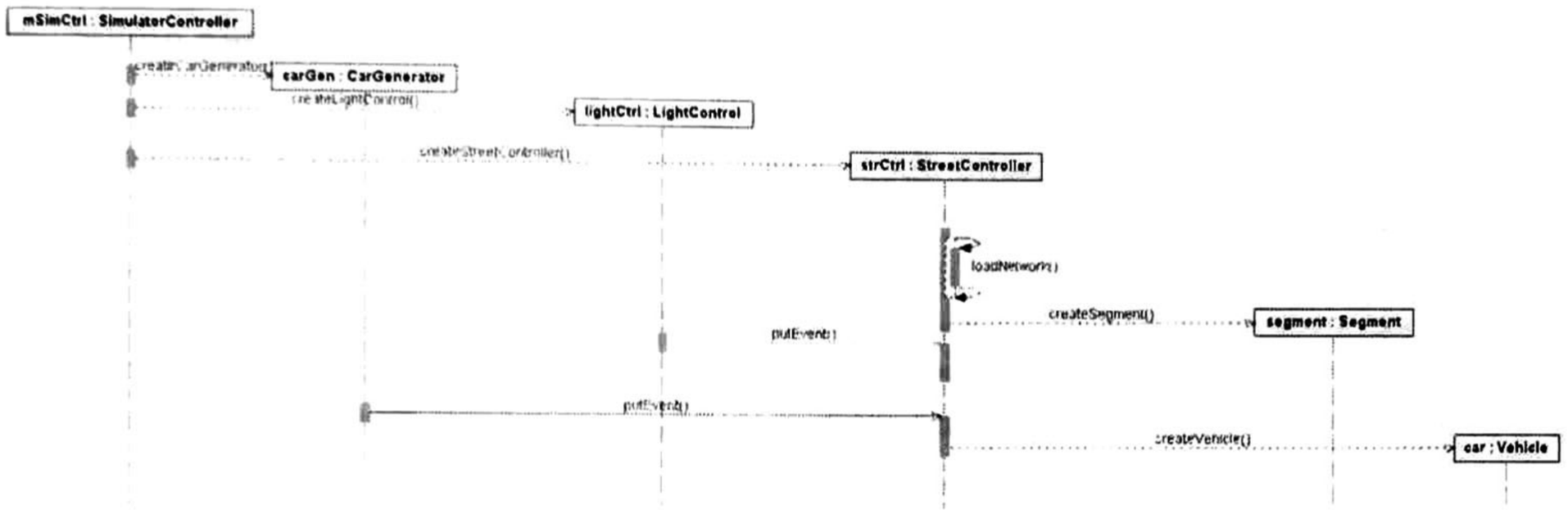


Fig. 4.9: Secuencia de inicialización de la simulación.

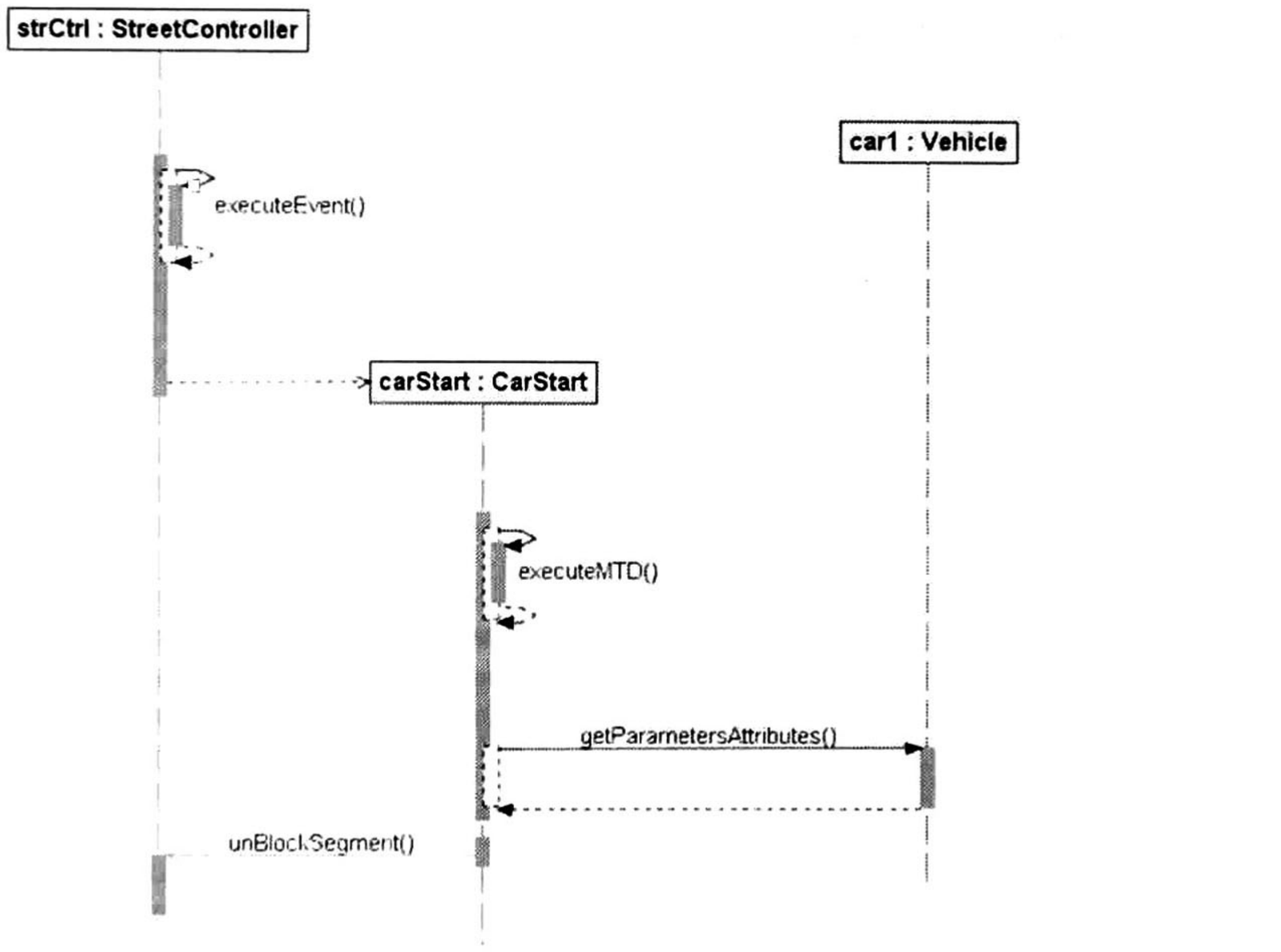


Fig. 4.10: Secuencia de ejecución de evento en la E^2F.

4.3.2. Máquina de simulación concurrente y distribuida

La implementación del algoritmo de ejecución concurrente y distribuida es presentada en el Algoritmo 4.1.

Algoritmo 4.1 Ejecución Maquina de simulación

```
1:  Inicializar las variables de estado ()
2:  Mientras no fin de la simulación Hacer
3           establecer t y deltaT
4           actualizar History
5:          Para cada ei de History hacer en paralelo
6                   Si hay conflicto Entonces
```

```
7:                          Repetir Until
8:                                  No existe más conflicto
9:                          endRepetir
10                  EndIf

    end if
```

Operación agregar evento (putEvent()).

Al inicializar la simulación es creado el nodo etiquetado como *rootControl*, este será el nodo encabezado de la lista de eventos potenciales, o la lista doblemente enlazada circular de eventos futuros ordenados de forma no decreciente. Inicialmente los campos *izq.* y *der* están referenciados a sí mismos. También son inicializados los *s* nodos etiquetados como *SQS*, donde *s*, es el número de divisiones en espacio (segmentos) de la estructura. Inicialmente los campos *sig.* y *ant*, hacen referencia a sí mismos.

Para realizar el encolamiento o inserción de un nuevo nodo (evento) a la estructura de implementación de la E^2F, se necesita especificar la lista *SQS* en la que será insertado (seleccionar el espacio), si este es el primer nodo de la estructura, el *rootControl* hará referencia a dicho evento así como el correspondiente *SQS* y se etiquetará como evento potencial. De no ser el caso, se busca en la lista encabezada por *SQS* el orden correcto y posteriormente si es un evento potencial se ubica en la lista encabezada por *rootControl* la posición correcta.

Algoritmo 4.2 Método para agregar un nuevo evento a la estructura de eventos pendientes en orden de ejecución (enqueue)

```
1·   Func: agregarEvento()
2.   set x = SQS
3    {x,y,z son variables auxiliares}
4    set z = null
5    set y = null
6    {evTime es el tiempo de ejecución del nuevo evento}
7    while (evTime >= x siguiente.evTime() && x.siguiente ¬ SQS ) do
8·       { buscar posición correcta en el SQS del segmento, brincando eventos
9·        del mismo tiempo dejando que sean los primeros en ser ejecutados}
10       set  x = x.siguiente
11:  end while
12   set y = x.siguiente
13.  set x.siguiente = new eventType()
14   set x.siguiente.schedule = evTime
15:  {Si es el primer evento en el segmento incluirlo en el círculo del rootControl}
16:  if ( x = SQS && y = SQS )then
17:      set z  = RootControl
18.      while(x.siguiente.getTime() >= z.derecho.getTime()) do
19·          set z = z.derecho
20:      end  while
21.      set  x.siguiente.derecho  =  z.derecho
22       set  x.siguiente.izquierdo  =  z
23       set  z.derecho.izquierdo  =  x.siguiente
24       set  z.derecho      x siguiente
25       set  x siguiente.eventoPotencial  =  true
26:  end if
27.  {si el evento a sustituir es potencial verificar la nueva posición de z en el círculo del rootControl}
28·  if (y.eventoPotencial = true) then
29.      set z = y
30:      while(x siguiente evTime() < z izquierdo.evTime && z.izquierdo ¬ rootControl) do
31·      set z = z izquierdo
```

```
32:         end while
33:     end if
34      {Colocarlo en la posición correcta de SQS}
35      set y anterior = x siguiente
36      set x siguiente anterior = x
37      set y anterior siguiente = y
38      {Quitar el antiguo potencial del circulo de rootControl agregar el Nuevo al circulo}
39      if( y eventoPotencial = true)then
40          set y eventoPotencial = false
41          set x siguiente eventoPotencial = true
42          set x = x siguiente
43          set x izquierdo = x izquierdo
44          set x izquierdo = x
45          set x izquierdo derecho = x
46          set x derecho = x
47          {elimina el antiguo del circulo de rootcontrol}
48          set y izquierdo derecho = y derecho
49          set y derecho izquierdo = y izquierdo
50          set y derecho = null
51          set y izquierdo = null
52:     end if
```

Operación sacar evento (executeEvent())

Para realizar la eliminación de algún nodo de la implementación de la E^2F, es necesario sacar el evento potencial que se encuentra al lado derecho del nodo etiquetado como *rootControl*. Si existen más eventos en la lista encabezada por el nodo etiquetado como *SQS*, entonces es necesario establecer el siguiente nodo como evento potencial, y buscar la correcta posición en la lista que tiene al nodo *rootControl* como encabezado.

Cuando el algoritmo selecciona sacar el evento siguiente a ejecutar, se bloquean los segmentos que forman una UNIDAD con este segmento (ver definición en el capítulo anterior), siguiendo las pre-condiciones y condiciones siguientes:

- Pre-condiciones:

Cada instancia de la clase *segment* contiene un evento etiquetado como *SQS*

La instancia de la clase *StreetController* tiene un evento etiquetado como *rootControl*

Cada instancia de la clase event tiene los campos

La instancia de la clase rootControl y cada una de las instancias de SQS, sus campos izq., der., sig., y ant. apuntarán a si mismos.

- Post-condiciones:

El siguiente evento a ser ejecutado es regresado y el evento que lo sustituirá es ordenado de acuerdo a su marca de tiempo y si es un evento potencial, con su atributo de *potentialEvent* puesto a verdadero.

El evento siguiente a ser ejecutado por la instancia de la clase unitSimulator (el evento con el menor marca de tiempo) es obtenido de la parte derecha del rootControl.

El último evento (el evento con mayor tiempo) es obtenido de la parte izquierda de la instancia *rootControl*.

Algoritmo 4.3 Método para sacar eventos (dequeue)

```
1:  Func: sacarEvento()
2:  set z = null
3:  set proximoEvento = rootControl.derecho
4   set x = rootControl.derecho.siguiente
5   {Busca en el círculo del rootControl ubicar el evento del circulo de SQS que quedo después de sacar el próximo evento}
6   if ( x ¬ SQS) then
7           set z = RootControl derecho
8           while ( x evTime() >= z derecho evTime() AND z.derecho ¬ rootControl) do
9           set z = z.derecho
10:         end while
11:         set x.eventoPotencial = true
12:         set x.izquierdo = z
13:         set z.derecho.izquierdo = x
14          set x.derecho = z.derecho
15:         set z.derecho =x
16:  end if
17  {Busca en el circulo de SQS}
18: set rootControl.setTime(proximoEvento.evTime())
19  set proximoEvento.anterior.setTime(proximoEvento.EvTime())
20  set proximoEvento.izquierdo.derecho = proximoEvento.derecho
21  set proximoEvento.derecho.izquierdo = proximoEvento.izquierdo
22: set proximoEvento.anterior.siguiente = proximoEvento.siguiente
23. set proximoEvento.siguiente.anterior = proximoEvento.anterior
24· set proximoEvento.anterior = null
25· set proximoEvento.siguiente = null
26. set proximoEvento.eventoPotencial = false
27: set proximoEvento.derecho = null
28: set proximoEvento.izquierdo = null
29.

31: proximoEvento.blockConflictSegments()
32.
33: return proximoEvento
```

4.4. Modelo de datos

La simulación utiliza un archivo en texto plano para cargar información sobre la red vial que se utiliza como escenario de simulación. Este archivo corresponde a la descripción estática de todos los segmentos de la red vial usando el lenguaje para la especificación del modelo microscópico del STU (UTYiL, por sus siglas en ingles). La característica formal del lenguaje permite sea automatizado y por lo tanto disminuye los errores de captura del modelo. El lenguaje propuesto permite capturar la información geográfica, topológica y de tránsito de un STU, permitiendo al ingeniero de tráfico modelar con gran nivel de detalle (microscópico) la sección de una ciudad en particular. El lenguaje está basado en el formato de marcado extensible (XML de las siglas en inglés), esto permite el almacenamiento de grandes cantidades de información y sigue la tendencia para el almacenamiento de mapas geográficos, tal como el proyecto LandXML, el cual es un estándar basado en XML que está siendo utilizado en las aplicaciones de ingeniería civil y GIS [15]. Además, se muestra como la información capturada por el lenguaje propuesto puede ser utilizada en otros modelos o simuladores como un meta-lenguaje.

4.4.1. Descripción de la estructura de red vial

En la Fig. 4.11 se muestra la descripción de un modelo completo de STU utilizando componentes del lenguaje UTYiL, los cuales son: *Segment, Object* y *Connection*. Esta representación jerárquica es resultado de la descripción formal presentada en el capítulo 2.

Etiqueta objetos

Cada calle en la ciudad cuenta con ciertos señalamientos que informan al usuario sobre: velocidad máxima, dirección de flujo, cruce peatonal, etc. Estos señalamientos son estáticos y no cambian o se mueven.

La regla sintáctica para capturar los objetos en UTYiL es definida por:

OBJECT:= <object id=[REF] type=[SIGNTYPE] pos=[POS] value=[VALUE]</>

Donde:

- *SIGNTYPE* puede ser cualquier valor que indique el tipo de señalamiento tal como: *bump* (tope), *depression*, *crossing*, *saw*, *stop*, *school*, etc.
- *POS* es la posición del objeto relativa al segmento
- *VALUE* es el valor adicional al señalamiento.

```
<?xml version="1.0" encoding=UTF-8"?>
<UTYiL>
    <network>
        <segments>
            <object>
            </object>

        </segments>
        <connections>

        </connections>
    </network>
</UTYiL>
```

Fig. 4.11:Estructura jerárquica y componentes del lenguaje UTYiL en un documento XML

Por ejemplo, cuando se requiere diseñar un plan de aquietamiento de tránsito, es decir, disminuir la velocidad de los vehículos al transitar por dichos segmentos colocando elementos físicos, tales como: topes, vibradores, cruces de peatones elevados, etc. son capturados como objetos en el modelo del STU.

En la Fig. 4.12 se muestra un ejemplo de un plan de aquietamiento de tránsito: un tope en la posición 20, con una altura de 15 cms, que abarca ambos carriles. Aparte de éstos se tiene el objeto de dirección de flujo en los carriles. En el carril 1 va hacia el punto a_i y en el carril 2 hacia el punto b_i, es decir los puntos extremos de cada segmento. Por último, existen dos objetos más: los señalamientos 1A y 2A que indican un límite de velocidad y la advertencia de próximo tope.

Utilizando UTYiL, el modelo de STU de la fig. 5 es descrito por:

<Object1 type="bump" pos="0.0" value="0"/>
<Object2 type="direction" pos="0.0" value="nodeB"/>
<Object3 type="speedLimit" pos="30.0" value="15.0"/>
<Object4 type="warningBump" pos="15.0" value="15.0"/>

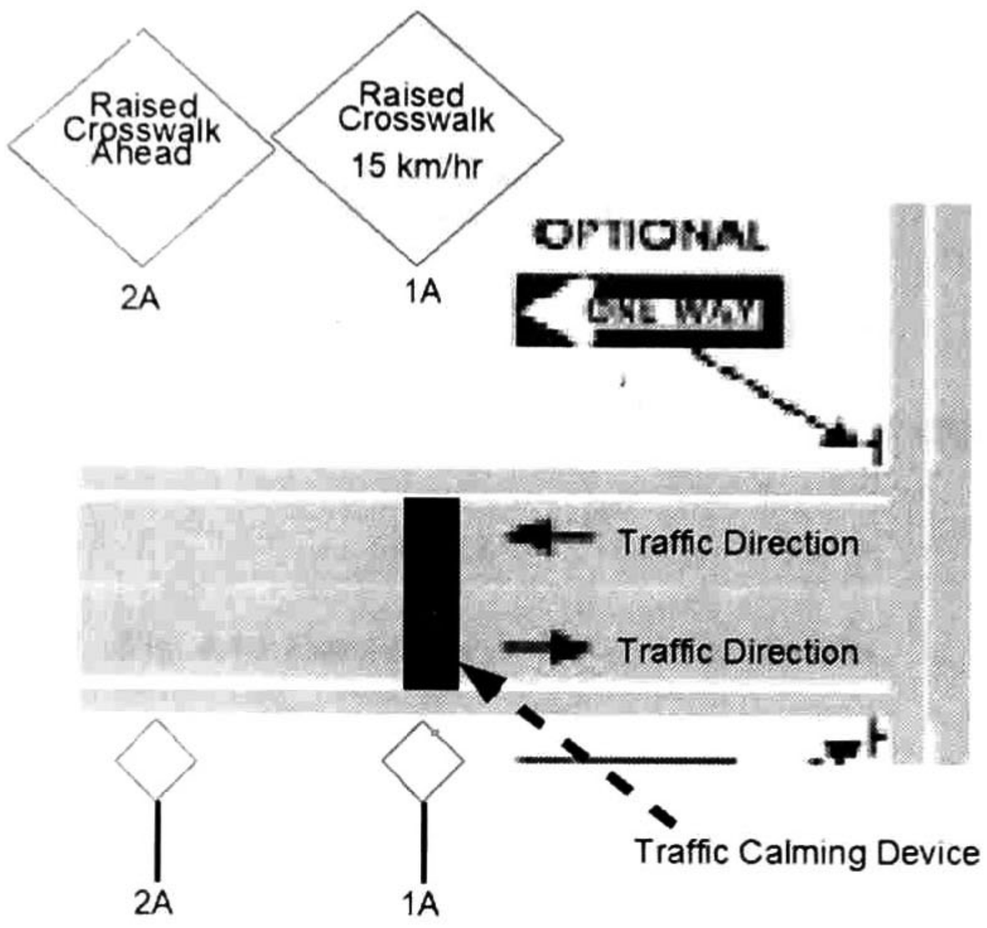


Fig. 4.12: Diseño de un plan de aquietamiento de tránsito

Etiqueta segmentos

Como se mostró en el capítulo 2, los segmentos representan solo un carril, y una calle es representada por *n* segmentos, donde *n* es el número de carriles de la calle.

La regla sintáctica que define a un segmento está formado por:

SEGMENT := id=[ID] segTy=[SEGMENTTYPE] x1=[X1] y1=[Y1] x2=[X2] y2=[Y2] [OBJECT|OBJECT OBJECT]

La longitud del segmento es obtenida a partir de los puntos x_1, *y1, x2, y2*.

La calle *"Escalona"* de la Fig. 4.13, es dividida en 3 segmentos (dicha calle tiene solo un carril), ya que la intersecan Valley y Breschka. Sin embargo, en el segmento 2 de la calle *Escalona*, existe un objeto, es un tope que necesita ser integrado al segmento.

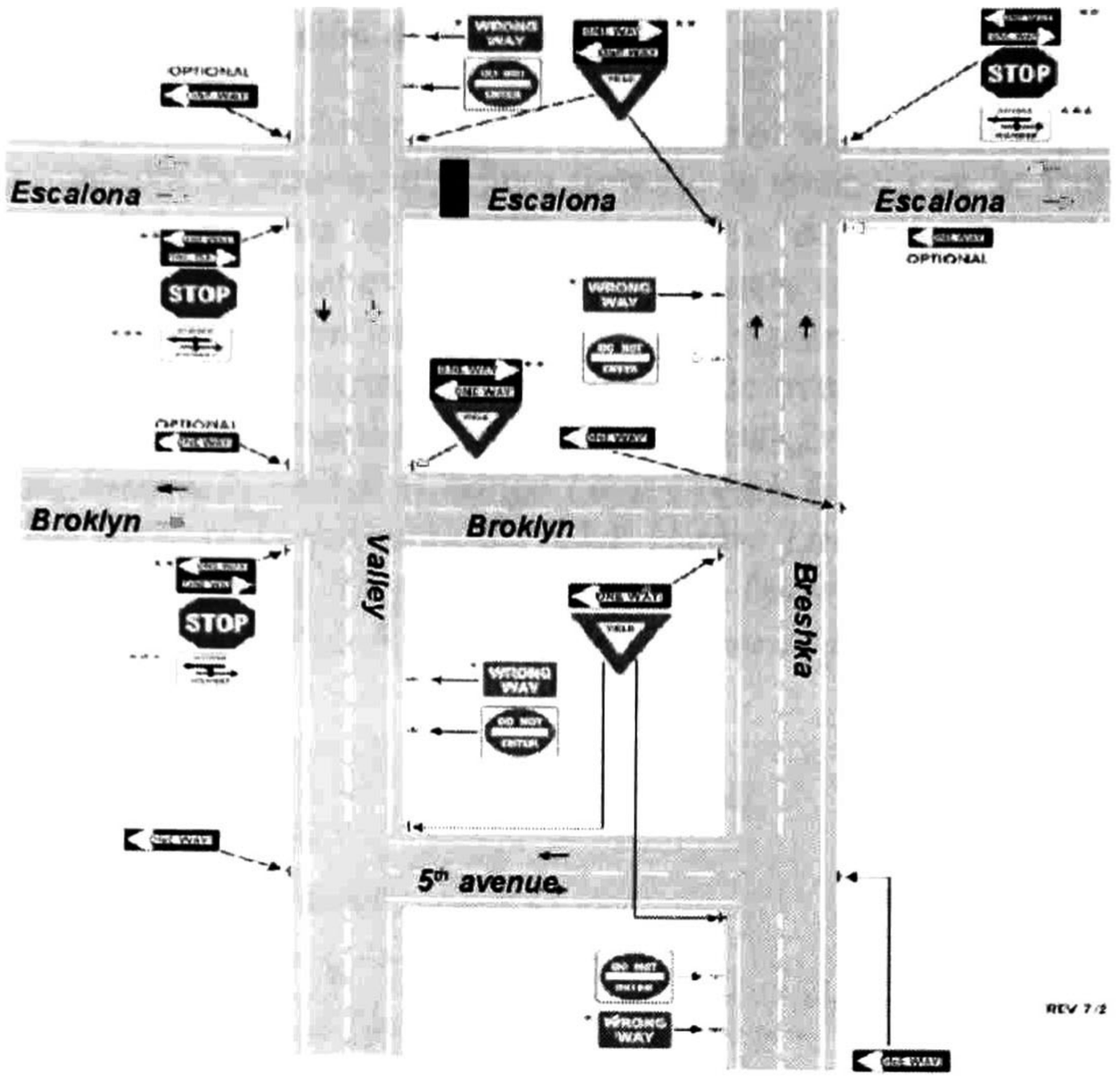


Fig. 4.13:Estructura de red con dos carriles

Entonces el ejemplo de la Fig. 4.13 quedaría capturada utilizando UTYiL como sigue:

```
<segment>
id="2" x1=20.0 y1=30.0 x2=20.0 y2= 35.0
<object id="1" type="bump" pos="0.0" value="0"/>
<object id="2" type="direction" pos="0.0" value="p2"/>
<object id="3" type="speedLimit" pos="30.0" value="15.0"/>
<object4 type="warningBump" pos="15.0" value="15.0"/>
</segment>
```

Etiqueta conexiones

La conexión entre los segmentos está definida por las relaciones *NS* y *NC* del apartado 3. En la Fig. 4.13 se muestra un segmento conectado de manera paralela al segmento 2 y secuencial con el segmento 3. Es traducida en el lenguaje de la siguiente manera:

```
<Connections>
        <connectionNS from ="1" to= "3"/>
        <connectionNC from ="1" to ="2"/>
</Connections>
```

4.4.2. Formato de ejecución de eventos

El simulador genera el archivo *log.dat*. Este archivo contiene el registro de los eventos ejecutados durante la simulación, cuyo formato se muestra en la Tabla 4.2. El primer campo (*objectID*) muestra el identificador único de cada vehículo; *evtime* contiene el tiempo de ejecución del evento; *segment* muestra la calle o segmento en el que se ejecutó el evento; *evPos* contiene la posición del segmento o calle en la que se ejecuto el evento y finalmente, *evtype*, contiene el tipo de evento que se ejecutó. Esta última tiene la siguiente interpretación: 1=**SE** (*Stop Event*),2=**CE** (*Cross Event*), 3 = **LCE** (*Light Change Event*), 5 = **CLE** (*Change Lane Event*), 6 = **CLEE** (*Change Lane End Event*), 7 = **ALE** (*Arrival Link Event*), 8 = **LLE** (*Leave Link Event*), 9 = **BE** (*Begin Event*), 10 = **AE** (*Arrival Event*), 11 = **WSE** (*Warning Event for stop*), 12 = **WBE** (*Warning Event for Start*). En la Fig. 4.14 se muestra el uso de algunos de estos eventos en la simulación.

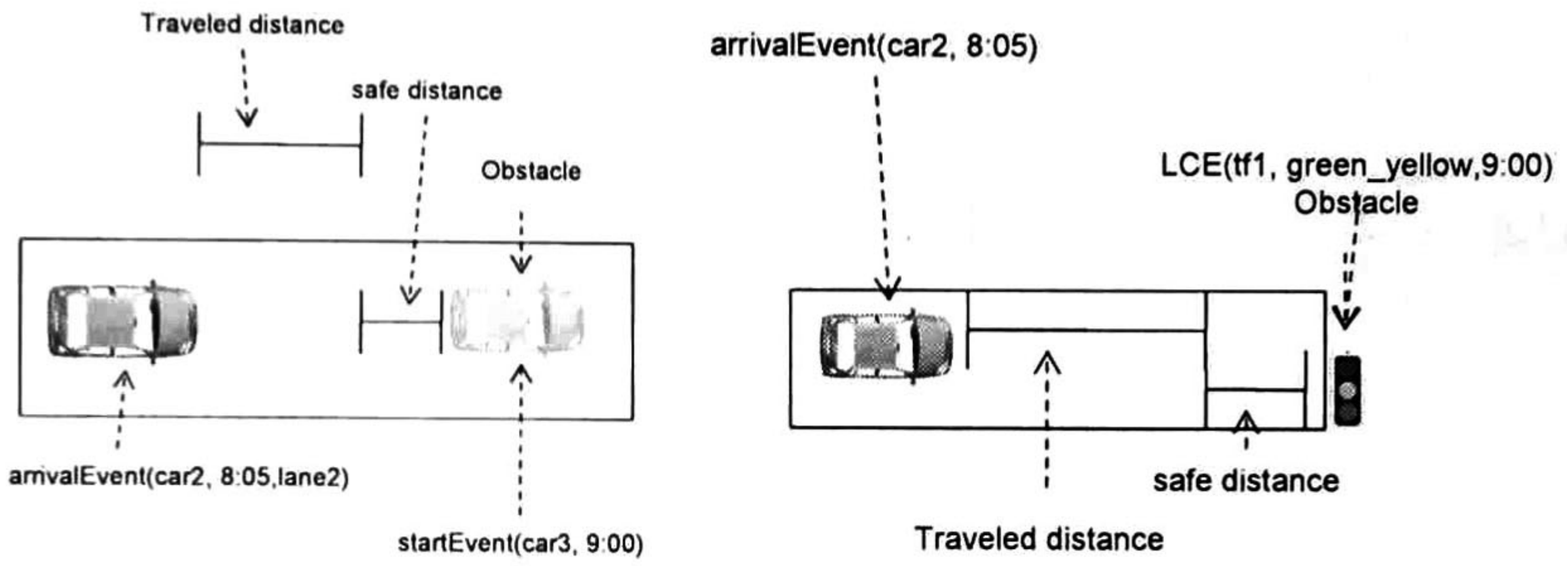


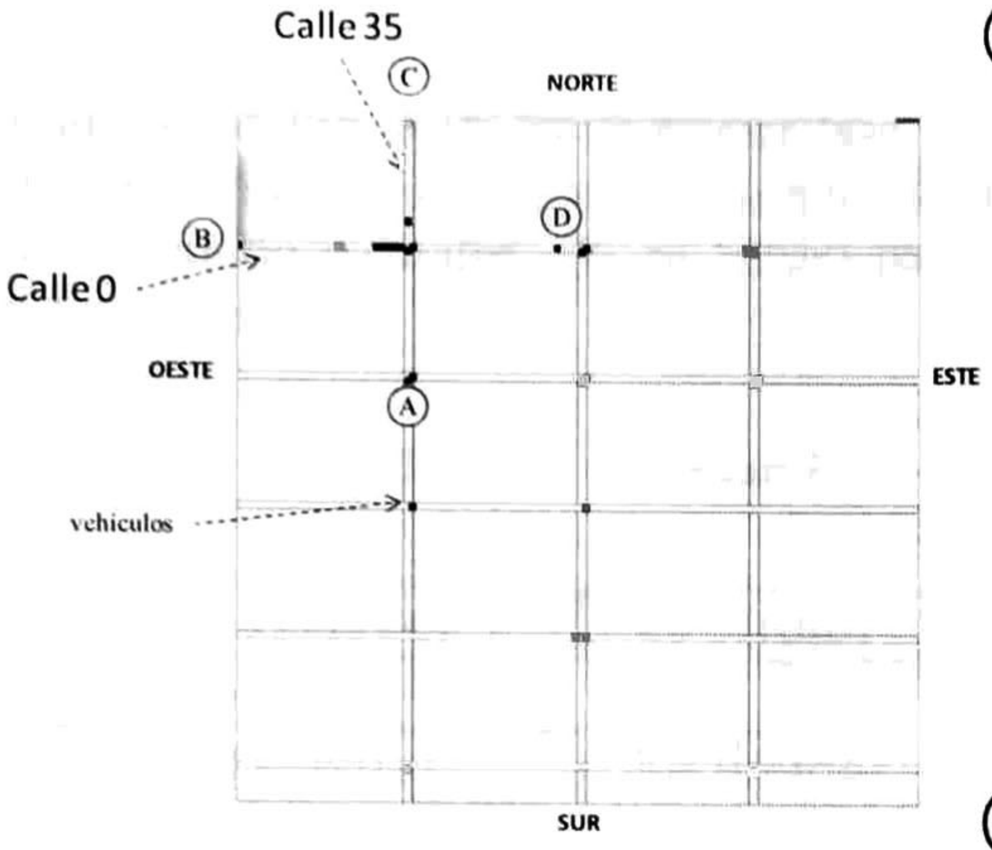
Fig. 4.14 Uso de los eventos en el simulador

objectID	evTime	segment	evPos	evType
25	55.36	20	0.0	7
23	55.36	23	124.0	11
25	55.36	20	132.0	8
23	55.36	23	124.0	12
39	55.47	10	132.0	2
39	55.47	15	0.0	7
12	55.59	25	128.0	11
14	55.59	28	128.0	11
12	55.59	25	128.0	12
14	55.59	28	128.0	12
58	55.87	9	0.0	7
58	55.87	4	132.0	2
12	55.95	25	132.0	2

Tabla 4.2: Datos de ejemplo, resultado de la simulación.

4.5. Caso de estudio

Las simulaciones realizadas se han orientado básicamente en la influencia de los semáforos sobre el tráfico a través de los distintos escenarios. El caso de estudio propuesto es de una sección urbana formada por 38 segmentos (calles de un solo carril) (ver Fig. 4.15). El tráfico es regulado por semáforos (tf_i) en cada intersección. Cada tf_i controla el flujo del segmento s_i.



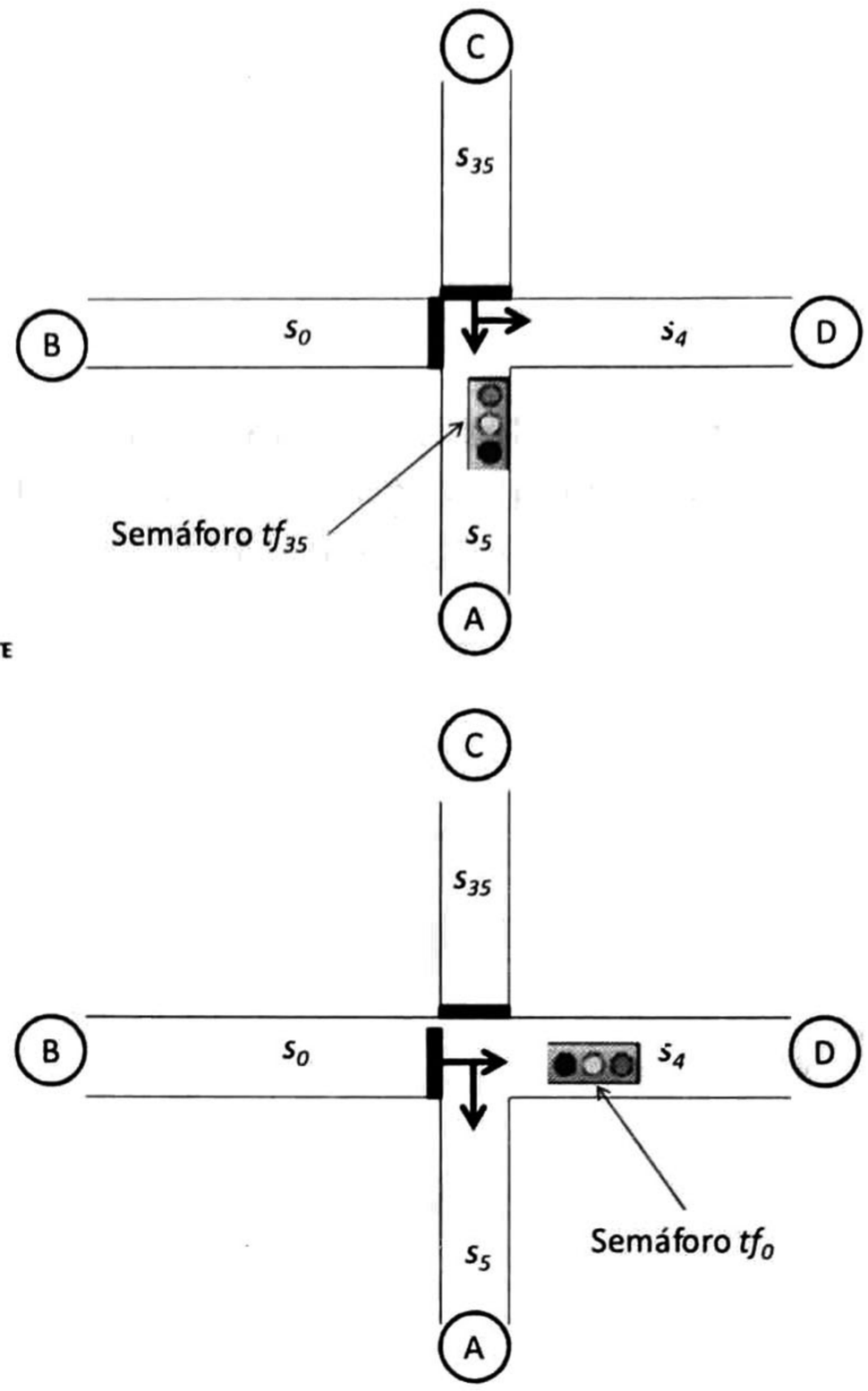


Fig. 4.15: Caso de Estudio, los semáforos correspondientes a la intersección de los segmentos s_{33} y s_0 y las fases de control implementadas en los semáforos

4.5.1. Generación de vehículos

El módulo de generación de vehículos requiere una lista de segmentos que funcionarán como segmentos fuente. En estos segmentos, el módulo de generación vehicular colocará los eventos para crear nuevos vehículos en la red. El modelo de generación de vehículos utilizado en este trabajo se basa en una función de distribución de *Poisson*, pero podría ser sustituida por un comportamiento real basado en una base de datos demográficos. En este caso, los segmentos fuente son los segmentos: 0, 1, 3, 35, 32, 34, 46 y 47. La distribución de *Poisson* es descrita por la fórmula:

$$P(n)=((\lambda t)^n e^{-\lambda t}) / n!$$

Donde:

- $P(n)$ es la probabilidad de que exactamente n vehículos lleguen al segmento durante t segundos.
- λ es la tasa de llegadas de vehículos (*veh/ min*)
- t es el intervalo durante el cual los vehículos son contados.

4.5.2. Control de tráfico

Para cada uno de los segmentos que confluyen en una intersección se les asigna un semáforo. Las estrategias de control utilizadas para el semáforo tf_0 (dirección oeste-este) y tf_{35} (norte-sur) durante la simulación se describen en la Fig. 4.16. En los primeros 600 segundos es usada la estrategia A y durante otros 600 segundos es usada la estrategia B. Las posibles rutas que un vehículo puede tomar en dicha intersección son mostradas en la Fig. 4.15. En la primer fase, el flujo es permitido (semáforo tf_{35} en verde) para los vehículos del segmento s_{35} que viajan de norte a sur. En la segunda fase el flujo es permitido (semáforo tf_0 en verde) para los vehículos del segmento s_0 que viajan de oeste a este.

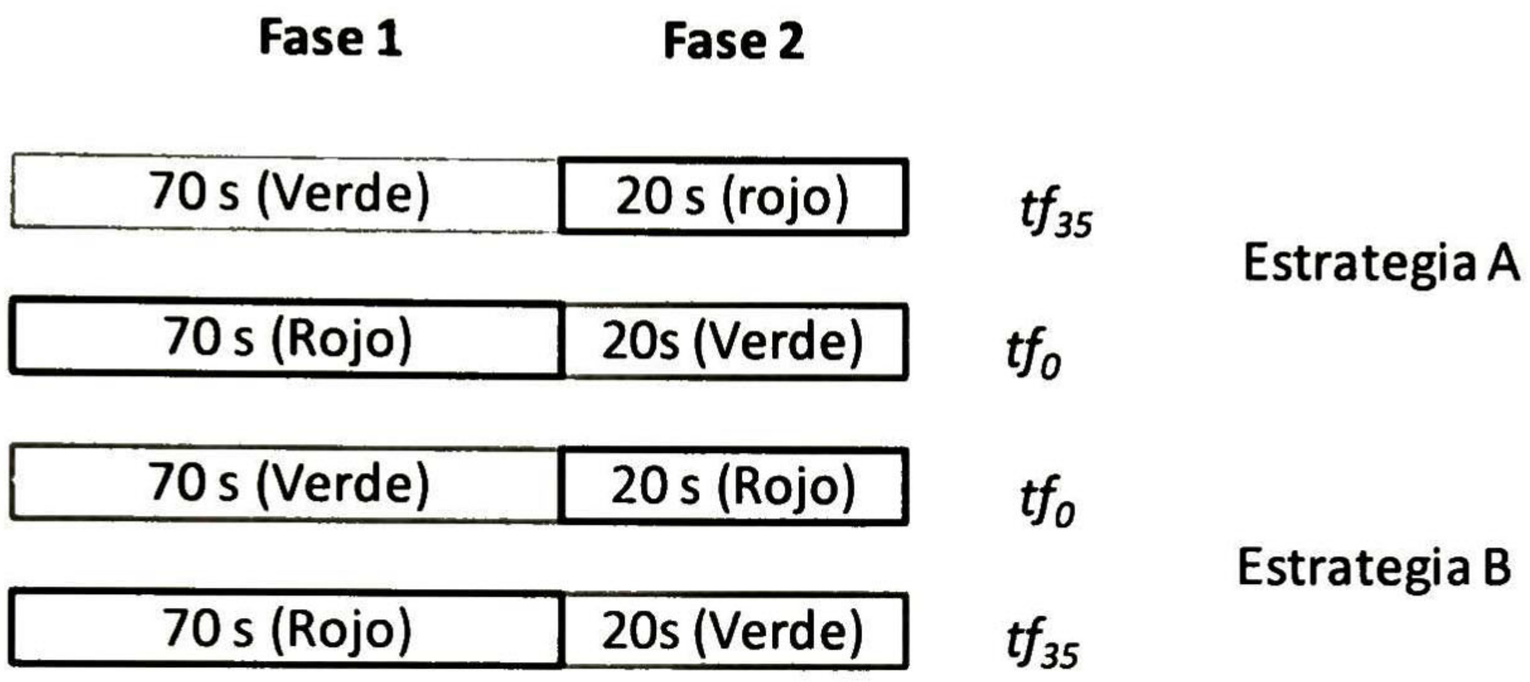


Fig. 4.16: Estrategias de control usadas en la simulación

4.5.3. Análisis de la simulación

El simulador cuenta con un modulo que analiza los resultados de la simulación contenidos en este archivo. Se obtiene el valor de la densidad y el flujo para el instante en el que se ejecutó el evento leído. Por ejemplo, en la Tabla 4.3 la primer línea es un evento de llegada de un vehículo al segmento 20, entonces se incrementa el número de vehículos que llegan; en la tercera línea, se lee un evento de salida del segmento, y se decrementa la variable de vehículos que salen del segmento.

Cada que se lee un evento del archivo, si es de tipo 7 (incrementa la variable *vehículos_que_han_entrado*) u 8 (incrementa la variable *vehículos_que_han_salido*), entonces se calcula el valor de la densidad y flujo en el segmento con las siguientes ecuaciones:

- **Flujo**: el número de vehículos que pasan por un punto por unidad de tiempo

Flujo = vehículos_que_han_salido (vehículos) / periodo_medicion (segundos)

- **Densidad**: el espacio es de un segmento, en esto caso es un valor constante de100 mts.

densidad = vehiculos_que_han_entrado – vehiculos_que_han_salido / segmento

Los valores de densidad y flujo obtenidos, se agregan a un archivo de salida llamado *flowdensity*.dat con el formato mostrado en la Tabla 4.3.

Densidad	Flujo
11.0	0.352941176470588 26
11.0	0.352941176470588 26
12.0	0.333333333333333
11.0	0.388888888888889
11.0	0.388888888888889
11.0	0.388888888888889
11.0	0.388888888888889
10.0	0.444444444444444
10.0	0.444444444444444
11.0	0.421052631578947 35
10.0	0.473684210526315 76
10.0	0.473684210526315 76
10.0	0.473684210526315 76
9.0	0.526315789473684 2
9.0	0.526315789473684 2
10.0	0.5
10.0	0.5
9.0	0.55

Tabla 4.3: Ejemplo de datos en el archivo log.dat de un segmento.

Usando los valores de este archivo se grafica en el eje de las *X* los valores de densidad, y en el eje de las *Y*, los valores de flujo (ver Fig. 4.17).

4.5.4. Análisis de flujo y densidad

Se grafican los resultados de la simulación, y se obtienen las gráficas densidad-flujo de la Fig. 4.17.

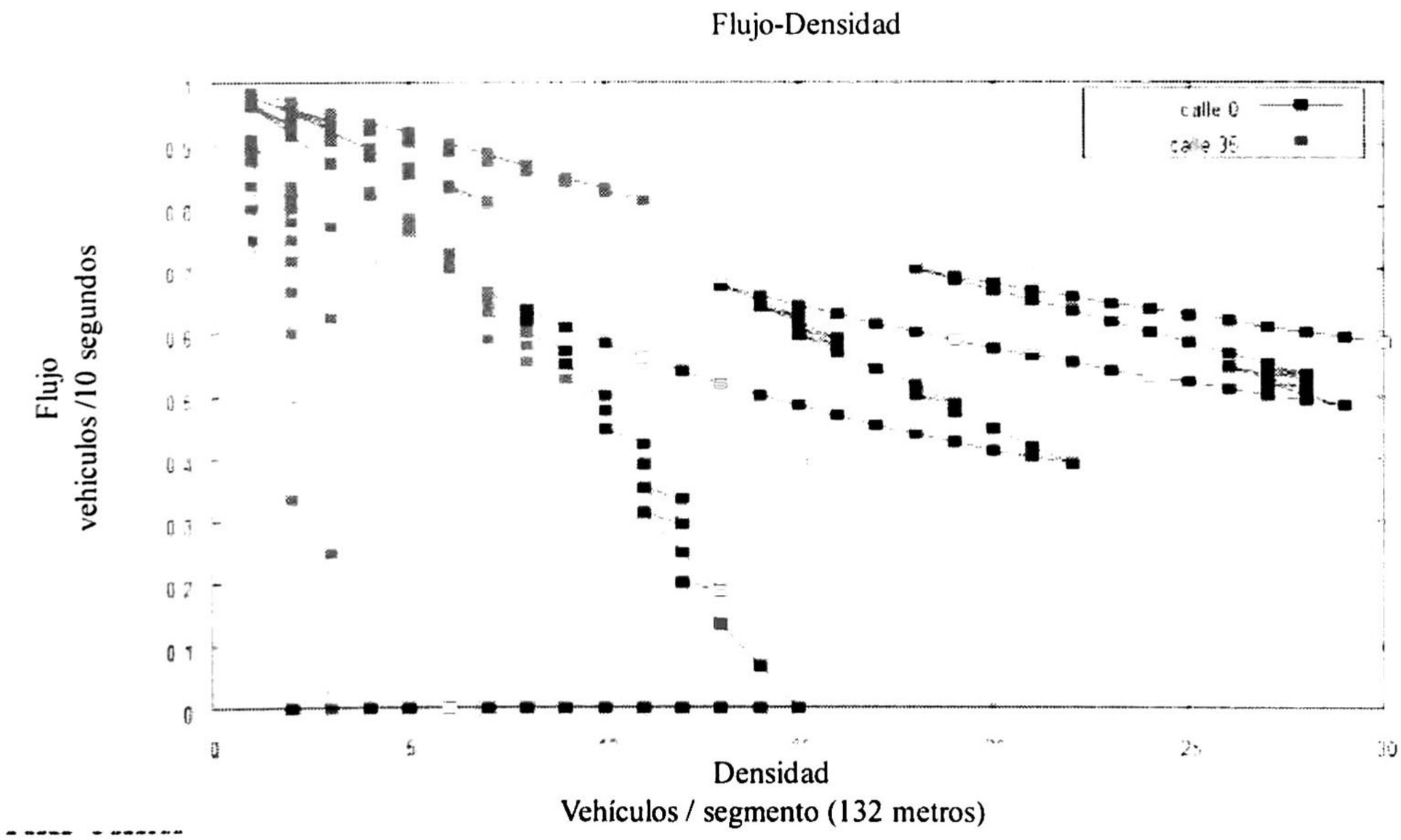


Fig. 4.17: Gráfica de Flujo-Densidad para el segmento 35 y el segmento 0

Obsérvese, como el segmento s_{35} mantiene mayores niveles de flujo (y por lo tanto menor densidad) que el segmento s_0, que debido a que desde el inicio se inicia a saturar, aunque existe el cambio a una fase que le da mayor prioridad no alcanza a liberar los vehículos que ya tiene acumulados. De hecho el segmento s_{35} nunca pasa de 15 vehículos por carril, lo contrario al segmento s_0, que va incrementando el número de vehículos en el carril (incrementando densidad). El diagrama de la Fig. 4.17 posee características del diagrama fundamental de flujo-densidad, esto es, a menor densidad mayor flujo y viceversa. Al estar cambiando de ciclo en el semáforo se está generando una nueva ola de vehículos, que se puede observar en el diagrama por la discontinuidad de la relación flujo-densidad. Esto va soportado con la visualización de la simulación, donde se pueden observar las colas (congestionamientos) del segmento s_0 y el estado libre de tráfico del segmento s_{35}.

4.5.5. Modificación de estrategias de control

La Fig. 4.18 muestra la relación flujo-densidad para el segmento s_0. El eje de las x muestra la densidad y el eje de las y el flujo. Observe como el diagrama fundamental flujo-densidad es conservado, esto significa, si la densidad incrementa su valor del flujo disminuye, como se observa en el área del elipse. La línea de "cambio de estrategia" indica el instante en el cual el semáforo cambia su estrategia de control. Cuando es usada la estrategia A, el tiempo de rojo es puesto a 70 segundos y el tiempo de verde a 20 segundos y cuando la estrategia B es seleccionada, el tiempo de rojo es puesto a 20 segundos y el tiempo de verde a 70 segundos. El flujo registrado con la estrategia B resulta ser mayor que cuando la estrategia A es usada.

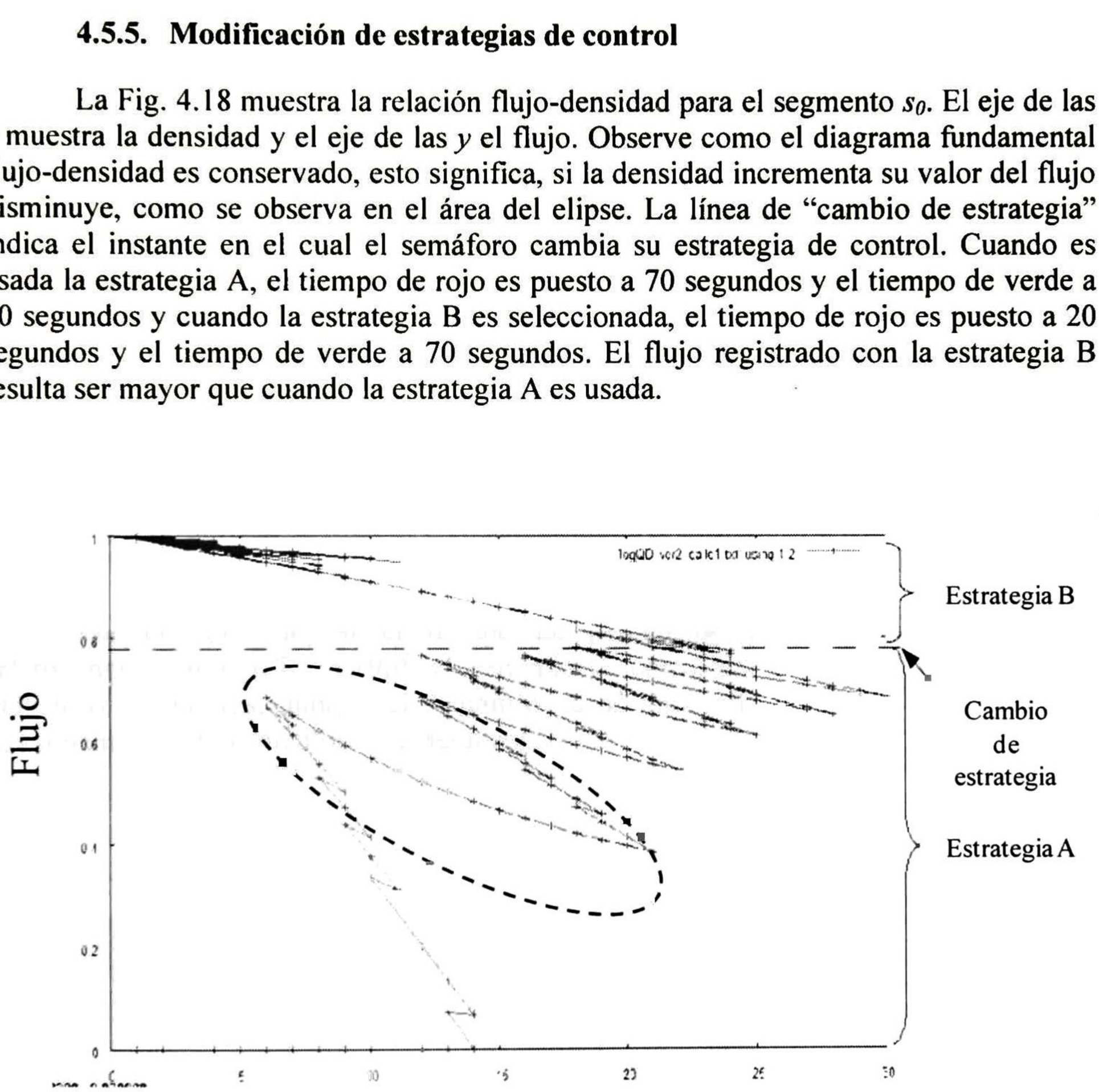


Fig. 4.18: Relación flujo-densidad en un segmento de la red

4.5.6. Modificación de parámetros individuales

El módulo generador de vehículos permite que se modifiquen los atributos de cada vehículo generado, tales como: velocidad preferida, distancia de seguridad entre vehículos, tiempo de percepción-reacción, etc. En la Fig. 4.19 se muestra la densidad de un segmento a durante un periodo de simulación con diferentes valores en el parámetro de tiempo de percepción-reacción del conductor. La densidad de la Fig. 4.19a se obtuvo con el parámetro tiempo de percepción-reacción del conductor igual a 0.2 min. El incremento en la densidad es causado por el acumulamiento de los vehículos al estar la luz de semáforo en rojo. Cuando la luz del semáforo cambia a verde, entonces se observa el pico de bajada, indicando el flujo de los vehículos por la intersección.

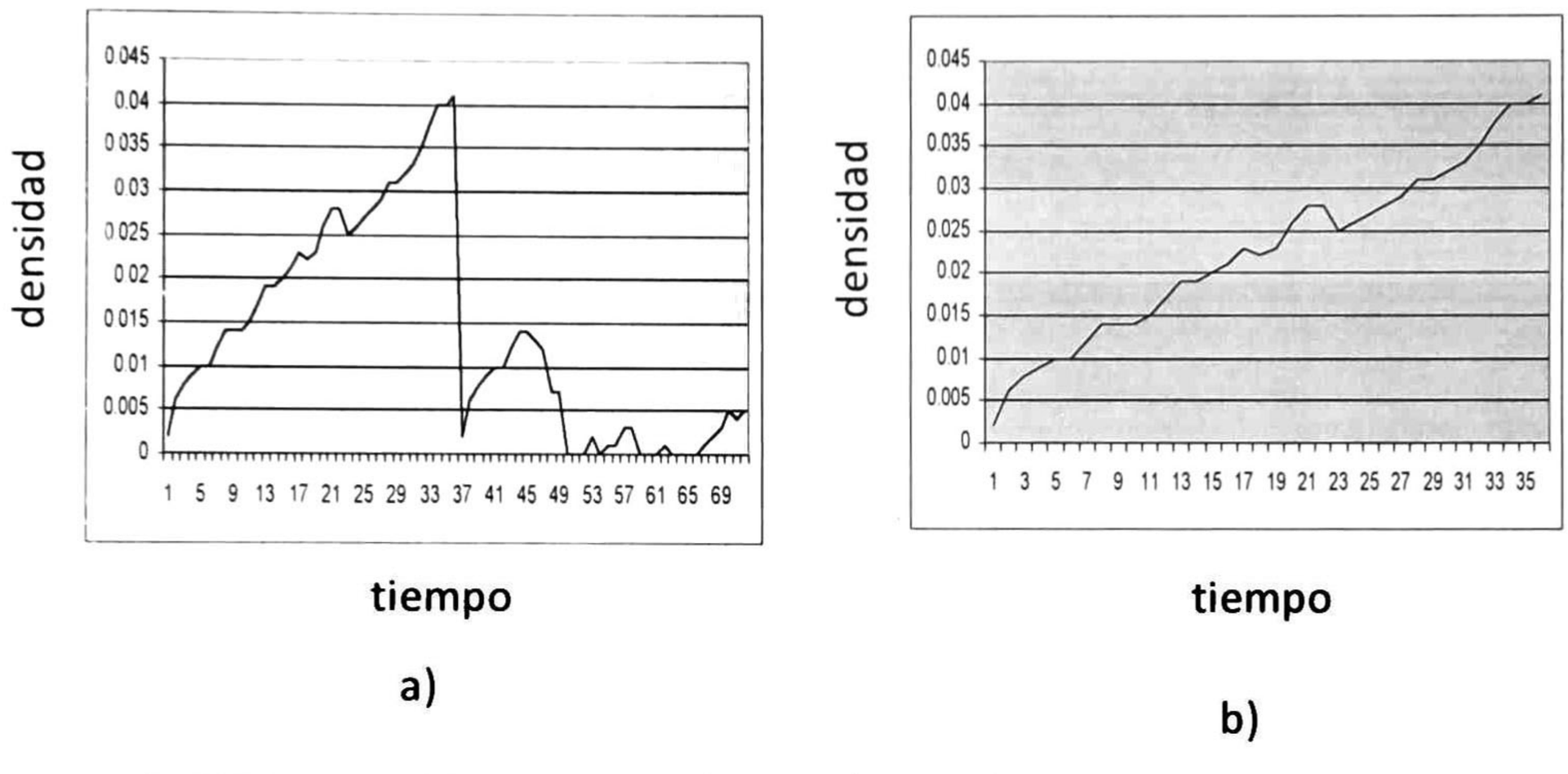


Fig. 4.19: Densidad del segmento con diferentes valores en el parámetro de retardo para arrancar a) un tiempo de percepción-reacción de 0.2 min y b) un tiempo de percepción –reacción de 8 min.

Por el contrario, si el tiempo de percepción-reacción del conductor es relativamente largo (Fig. 4.19b), el vehículo se convertirá en un obstáculo para los vehículos seguidores, aunque el semáforo ya se encuentre con luz verde, causando acumulamiento de vehículos en el segmento.

CONCLUSIONES Y LÍNEAS DE INVESTIGACIÓN FUTURAS

5.1. Conclusiónes

La micro-simulación de tráfico urbano es un tema vasto y complejo. En este trabajo se ha utilizado el enfoque de los sistemas de agentes móviles para manejar de una manera natural la descripción del comportamiento de los vehículos, que son las entidades que generan los eventos de la simulación. Consecuentemente, el manejo de los eventos en forma discreta coordina la ejecución colectiva de estas entidades que interactúan entre sí y con otras entidades de la red de tráfico; la distribución de esta coordinación se hace necesaria ante la gran cantidad de eventos.

En esta tesis se han abordado dos aspectos importantes de este problema como son la especificación de un STU y el diseño de un simulador distribuido; en cada uno de ellos se ha propuesto una solución que aventaja las propuestas anteriores en la materia. La implementación realizada permitió comprobar el funcionamiento de la metodología y arquitectura propuestas.

La metodología propuesta para especificar el STU está basada principalmente en el uso de nLNS. Un modelo de tres niveles permite capturar de una manera clara el comportamiento de los componentes de un STU y su interrelación; se obtiene una descripción modular y jerárquica de la red vial y las entidades dinámicas, cuya flexibilidad permite modificar fácilmente dicho modelo. Por otro lado la construcción de las bases de conocimiento para la toma de decisiones como reglas *if-then* es también flexible por su modularidad, permitiendo experimentar mecanismos de razonamiento más complejos. Estas características permiten introducir variantes fácilmente al modelo para analizar comportamientos y escenarios diversos.

La especificación obtenida tiene un alto grado de concurrencia dada la independencia de los modelos que describen el comportamiento de los vehículos, los cuales evolucionan dentro del modelo que describe la red de tráfico. Esta evolución genera una gran cantidad de eventos que son manejados a través de la arquitectura de simulación propuesta, la cual permite explotar al máximo el paralelismo del modelo.

El simulador desarrollado en base a la arquitectura hace uso de una biblioteca de clases, construida en Java, para implementar los componentes de la especificación y de un meta-lenguaje (UTYiL), basado en el formato XML, para la descripción de las características geográficas, topológicas y de tránsito de un STU. Las pruebas efectuadas con el simulador validaron la pertinencia del modelo de STU y la efectividad de la arquitectura.

Las contribuciones de esta tesis constituyen un paso adelante en el estudio de los STU por medio de microsimulación distribuida basada en agentes móviles. Se han abordado los temas más importantes del enfoque, mostrando su viabilidad. Existen naturalmente otros aspectos que deben abordarse y/o mejorarse en este estudio a fin de completar la metodología y arquitectura propuestas. A continuación se mencionan algunas de estas áreas de oportunidad.

- Introducir en la metodología de modelado los elementos necesarios para incluir otros agentes móviles, tales como vehículos con carril reservado y peatones. Igualmente es interesante la representación de eventos que bloquean temporalmente un carril, por ejemplo descomposturas de vehículo y colisiones por alcance.
- El mecanismo de coordinación de la ejecución distribuida del simulador puede ser mejorado utilizando un agente especializado que maneje la comunicación; esto permitiría acelerar la simulación.

- Introducir agentes fijos en cada crucero que se encarguen de controlar los semáforos; esto permitirá desarrollar políticas de control complejas en la que estos agentes se coordinen de manera inteligente.

La interface con el usuario merece desarrollarse en detalle. Aunque la introducción de la especificación y la visualización de los resultados son tareas independientes a las estrategias de simulación, una interfaz amigable (editor y despliegue) hace accesible este tipo de simuladores a usuarios que no son especialistas en urbanismo.

LISTA DE PUBLICACIONES DERIVADAS DE LA TESIS

- Conferencias

— E. Lopez-Neri, E. Lopez-Mellado, and A. Ramirez-Treviño, "Un lenguaje para la descripción de la información geográfica de sistemas de tráfico urbano," in *in IV Semana Nacional de Ingeniería Electrónica* , Aguascalientes,Ags., 2008.

— E. Lopez-Neri, E. Lopez-Mellado, and A. Ramirez-Trevino, "Hierarchical Modeling of Urban Traffic systems for Multi-Agent based simulation," in *in International conference on Modeling, simulation and control 2008*, San Francisco, California, 2008a.

— E. Lopez-Neri, E. Lopez-Mellado, and A. Ramirez-Treviño, "Microscopic Modeling Framework for Urban Traffic Systems Simulation," in *the 7th International conference on system simulation and scientific computing*, Beijing, China, 2008.

— J. Trejo-Sánchez, E. Lopez-Neri, E. López-Mellado, and A. Ramírez-Treviño, "Multi-Agent Based Urban Traffic Control," in *In Procc. Of the 13th International Congress on Computer Science Research CIICC'06*, Tampico, México., 2006.

— E. López-Neri, E. Lopez-Mellado, and A. Ramirez-Treviño,"Modelado y simulación de sistemas multi-agentes orientados a eventos," in *V Semana Nacional de Ingeniería Electrónica* , Ocotlán, Jalisco, México., 2009.

- Revista

— E. López-Neri, Ramirez-Treviño, Antonio, López-Mellado, and Ernesto, "A modelling framework for urban traffic systems microscopic simulation," *Simulation Modelling Practice and Theory*, p. 17, 2009. doi:10.1016/j.simpat.2009.09.007

6. Referencias

[1] ECO. (2007,) Canada's New Government announces targets to tackle climate change and reduce air pollution. [Online]. http://www.ecoaction.gc.ca/news-nouvelles/20070426-7-eng.cfm

[2] CEC, "European transport Policy for 2020: time to decide. „" Com (2001) 370, 2001.

[3] OECD, "Road Transport Research: Outlook 2000 – Perspectives 2000," 2000.

[4] J. Barceló, J. Casas, J. L. Ferrer, and D. Garcia, "Modelling Advanced Transport Telematic Applications with Microscopic Simulators: The case of AIMSUN2"," in *Traffic and Mobility, Simulation, Economics, Environment*, 1999.

[5] Q. Yang, "A simulation laboratory for evaluation of dynamic traffic management systems," 1997.

[6] R. R. Rosales, "Simplest Car Following Traffic Flow Model," in *MIT, March 26*, 1999.

[7] R. W. Rothery, "Car Following Models," in *Traffic Flow Theory*. Washington D.C.: Transportation Research Board, Special Report No. 165, 1975, ch. 4, pp. 4-1,4-41.

[8] K. Nagel and M. Schreckenberg, "A cellular automaton model for freeway traffic," *Journal of Physique I*, vol. 2, pp. 2221-2229, 1992.

[9] D. Chowdhury, L. Santen, and A. Schadschneider, "Statistical Physics of Vehicular Traffic and Some related Systems,' *In Physics Reports*, vol. 329(4-6), pp. 199-329, 2000.

[10] A. Hamalainen, "Studies of traffic situations using cellular automata," PhD. Dissertation, Helsinki University of Technology, Espoo, Finland, 2006.

[11] M. Mahut, "A Discrete Flow Model for Dynamic Network Loading," 2001.

[12] P. A. M. Ehlert, "Intelligent Driving Agents: The agent approach to tactical driving in autonomous vehicles and traffic simulation," 2001.

[13] J. M. Leitao, A. A. Sousa, and F. N. Ferreira, "Graphical control of autonomous, virtual vehicles," in *Proceedings of the IEEE Vehicular Technology Conference*, Tokyo,Japan, 2000, pp. 507-511.

[14] P. Ferreira, E. Esteves, R. Rossetti, and E. Oliveira, "Extending microscopic traffic modelling with the concept of situated agents," in *The 5th Workshop on Agents in Traffic and Transportation, 7th International Conference on Autonomous Agents and Multi-Agent Systems*, Estoril, Portugal, 2008.

[15] S. El Hadouaj, S. Espie, and A. Drogoul, "A multi-agent Road Traffic Simulation Model: Validation of the Insertion Case," in *Proceedings of 2004 Summer Computer Simulation Conference (SCSC'04),*, San Jose, California, USA, 2004.

[16] B. Lacroix, et al., "Towards traffic generation with individual driver behavior model based vehicles," in *Proceedings of the Driving Simulation Conference North America*, 2007, pp. 144-154.

[17] A. Champion, R. Mandiau, C. Kolski, A. Heidet, and A. Kemeny, "Traffic generation with the SCANeR II simulator: towards a multi-agent architecture,' in

Proceedings of the Driving Simulation Conference, 1999, pp. 311-324.

[18] T. Schulze and T. Fliess, "Urban Traffic Simulation with psycho-physical vehicle-following models," in *Proceedgins of the 1997 Winter simulation Conference*, 1997.

[19] G. W. Russell, N. S. Ferguson, P. Shaw, G. J. Milne, and J. McInnes, "The rapid simulation of urban traffic using field programmable gate arrays," in *International conference on Application of New Technology to Transport Systems*, vol. 1, 1995, pp. 107-122.

[20] W. J. Barclay, "Point-to-Point microscopic simulation: a discussion of issues," in *Proceedings of the First Western Pacific and Third Australia-Japan Workshop on stochastic models*, Christchurch, New Zealand, 1999.

[21] K. Nagel, D. Charypar, and K. W. Axhausen, "An event-driven parallel queue-based microsimulation for large scale traffic scenarios," in *ETH, Eidgenössische Technische Hochschule Zürich, IVT*, 2007.

[22] D. A. Roozemond and J. L. H. Rogier, "Agent controlled traffic lights," in *ESIT 2000*, Aachen,Germany, 2000, pp. 77-82.

[23] Z. Liu, "A survey of intelligence methods in urban traffic signal control," *IJCSNS International Journal of Computer Science and Network Security*, vol. 7, no. 7, pp. 105-112, Jul. 2007.

[24] A. L. C. Bazzan, D. de Oliveira, F. Klugl, and K. Nagel, "To adapt or not to adapt Consequences of adapting driver and traffic light agents," in *Adaptive Agentes and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*. Springer Berlin/ Heidelberg, 2008, pp. 1-14.

[25] P Waddell and G. F. Ulfarsson, "Introduction to Urban Simulation: Design and Development of Operational Models," in *Handbook in Transport*, b. K. H. e. Stopher, Ed. Pergamon Press, 2004, vol. 5: Transport Geography and Spatial Systems, pp. 203-236.

[26] W. J. Upton. (2003) Oregon modeling improvement. [Online]. http://tmip.fhwa.dot.gov/clearinghouse/case_studies/oregon_modeling.stm

[27] M. H. Lighthill and G. B. Whitham, "On kinematic waves II: A theory of traffic flow on long crowded roads," in *Proc. Royal Soc.*, London, 1955, pp. 317-345.

[28] C. Zocchi, C. Rovetta, and F. Fanfulla, "Physiological parameters variation during driving simulations," in *International conference on Advanced Intelligent Mechatronics*, 2007, pp. 1-6.

[29] J. Colomer, "Planificación y Economía del Transporte," Universidad Pontificia de Valencia, Valencia, España, Reporte, 1998.

[30] B. R. Izquierdo, "Los modelos de transporte," in *Modelos de Respuesta Rápida en Distribución Física de Mercancias*, J. V. Colomer, Ed. Universidad de Cantabria, 1995.

[31] M. Ghanbari, C. J. Hughes, M. C. Sinclair, and J. P. Eade, *Principles of Performance Engineering for Telecommunication and Information Systems*. 1997.

[32] M. Mccartney and M. Carey, "Follow that car!, Investigating a simple class of car following model,' *Teaching Mathematicas and Its Applications*, vol. 19, no. 2, pp. 83-87, 2000.

[33] P. Mussi, T. Jiménez, and G. Siegel, "A road traffic simulator: car-following and lane-changing," in *European Simulation MultiConference 200*, Gent, Belgium,

2000, pp. 24-245.

[34] F. Wang, et al., "A new car-following model inspired by Galton board,' arXiv:0807.3081 [physics.data-an], 2008.

[35] R. Wilson, "An analysis of Gipp's car-following model of highway traffic ," *IMA Journal of Applied Mathematics*, p. 316, 2001.

[36] P. G. Barros, J. Kelner, R. C. Farias, and D. A. Pessoa, "Three-dimensional urban traffic simulation with ITrans," in *VIII Symposium on Virtual Reality,SVR 2006*, Belém, Pará, Brazil, 2006.

[37] S. Lei, J. Hanping, and L. Huapu, "Research of urban microscopic traffic simulation system," in *Proceedings of the Eastern Asia Society for Transportation Studies*, 2005, pp. 1610-1614.

[38] P. Hidas, "Modelling Lane Changing and Merging in Microscopic Traffic Simulation,' *Transportation Research, Part C: Emerging Technologies*, vol. 10, no. 5-6, pp. 351-371, Oct. 2002.

[39] P G. Gipps, "A behavioural car-following model for computer simulation," *Transportation Researc*, vol. 15B, pp. 105-111, 1981.

[40] J. Barcelo and J. Casas, "Dynamic network simulation with AIMSUN," in *Proceedings of the Internation Symposium on Transportation*, 2002.

[41] K. S. Perumalla, "A Systems Approach to Scalable Transportation Network Modeling," in *proceedings of the 2006 Winter Simulation Conference*, vol. 0, 2006, pp. 1500-1507.

[42] K. S. Perumalla and B. L. Bhaduri, "On accounting for the interplay of kinetic and non-kinetic aspects in population mobility models," in *European Modeling and Simulation Symposium*, 2006.

[43] T. Sun and J. Wang, "A traffic cellular automata model based on road network grids and its spatial and temporal resolution s influences on simulation," *Simulation Modelling Practice and Theory*, vol. 15, pp. 864-878, 2007.

[44] K. Nagel and P. Stretz. (1997) TRANSIMS traffic flow characteristics. [Online]. http://arxiv.org/abs/adap-org/9710003

[45] E. G. Camparil and G. Levi, "A cellular automata model for highway traffic," *Eur. Phys. J.* . vol. B, no. 17, pp. 159-166, 2000.

[46] N. Cetin, A. Burri, and K. Nagel, "A Large-scale agent-based traffic microsimulation based on queue Model,' in *3rd Swiss Transport Research Conference STRC'03*, Monte Verita, 2003.

[47] J. Wang, C. Jin, and Y. Deng, "Performance analysis of traffic networks based on stochastic timed Petri net Models," in *Proceedings of the 5th IEEE International conference on Engineering of Complex Computer Systems*, Las Vegas, 1999, pp. 77-85.

[48] C. Tolba, D. Lefebre, P. Thomas, R. Bouyekhf, and A. ElMoudni, "Continuous Petri nets for the microscopic modeling of traffic flow," in *Proceedings of the SCSC'02*, San Diego, 2002.

[49] F. Di Cesare, P. T. Kulp, M. Gile, and G. F. List, "The application of Petri nets to the modeling, analysis and control of intelligent urban traffic networks," in *Lecture Notes in Computer Science*, 1994.

[50] A. Di Febbraro and N. Sacco, "On modeling urban transportation networks via hybrid Petri nets,' *Control Engineering Practice*, vol. 12, no. 10, pp. 1225-1239,

2004.

[51] J. Julvez and R. Boel, "Modelling and controlling traffic behaviour with continuous Petri nets," in *Proceedings of the 16th triennial world congress of the International Federation of Automatic Control (IFAC2005)*, Prague, Czech Republic, 2005.

[52] T. R. Boon-Leng, "A study to model human behavior in discrete event simulation (DES) using SIMKIT," Master´s Thesis, Naval Postgraduate School, Monterey, California, 2007.

[53] Cicortas and N. Somosi, "Multi-Agent system model for urban traffic simulation," in *SACI'05: 2nd. Romanian-Hungarian Joint Symposium on Applied Computational Intelligence*, 2005.

[54] E. Bonebeau, "Agent-based modeling: Methods and techniques to simulate human systems," *PNAS*, vol. 99, no. 3, pp. 7280-7287, 2002.

[55] R. Rossetti, E. Oliveira, and A. Bazzan, "Towards a specification of a framework for sustainable transportation analysis," in *Workshop on Artificial Intelligence Applied to Sustainable Transportation Systems, 13th. Portuguese Conference on Artificial Intelligence.*, Guimaraes, 2007.

[56] H. S. Sudhira, "Integration of agent-based and cellular automata models for simulating urban sprawl,' MSc in Geo-informatics, International Institute for Geo-information Science and Earth Observation Enschede, The Netherlands, 2004.

[57] P. A. F. Ferreira, E. F. Esteves, R. J. F. Rossetti, and E. C. Oliveira, "Applying situated agents to microscopic traffic modelling," in *Multi-Agent Systems for Traffic and Transportation*, A. L. C. Bazzan and F. Kluegl, Eds. PA:IGI Global (to appear), 2009.

[58] P. Ferreira, E. Esteves, R. Rossetti, and E. Oliveira, "A cooperative simulation framework for traffic and transportation engineering," in *Cooperative Design, Visualization, and Engineering, 5th. International Conference, CDVE 2008*, L. Yuhua, Ed. Calvia, Mallorca, Spain: Lecture Notes in Computer Science 5220 Springer 2008, 2008, pp. 89-97.

[59] P. A. Fonseca Ferrerira, "Specification and implementation of an artificial transport system," Faculdade de Engenharia da Universidade do Porto (FEUP), Report of Dissertation Master in Informatics and Computing Engineering, 2008.

[60] R. T. V. Katwijk, P. V. Koningsbruggen, B. D. Schutter, and J. Hellendoorn, "A test bed for multi-agent control systems in road traffic management," in *Proceedings of the 84th Annual Meeting of the Transportation Research Board*, Washington, DC, 2005, p. 22.

[61] K. Bouchefra, R. Reynaud, and T. Maurin, "IVHS viewed from a multiagent approach point of view," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, Piscataway, 1995, pp. 113-117.

[62] P. Davidsson, L. Henesey, L. Ramstedt, J. Tornquist, and F. Wernstedt, "An analysis of agent-based approaches to transport logistics," *Transportation Research Part C*, pp. 255-271, 2005.

[63] E. Oliveira and N. Duarte, "Making way for emergency vehicles," in *European Simulation and Modelling Conference*, Ghent, 2005, pp. 128-135.

[64] (2008, Jul.) Paramics: Microscopic Traffic Simulation developed by Quadstone Ltd. . [Online]. www.paramics-online.com

[65] (2008, May) Jade: Java Agent Development Framework. [Online]. http://jade.tilab.com/

[66] (2008, Aug.) Jess: The rule engine for the Java platform. [Online]. http://www.jessrules.com/

[67] A. Gourgoulis, P. Kacsuk, G. Terstyansky, and S. Winter, "Using clusters for traffic simulation,' in *Proceedings of the 26th. International Conference on Hypermedia and Grid Systems (HGS), MIPRO2003*, Opatija, Croatia., 2003, pp. 123-128.

[68] T. Delaitre, et al., "Traffic simulation in P-Grade as a grid service," in *Proceedings of the DAPSYS 2004 Conference*, Budapest, Hungary, 2004, pp. 129-136.

[69] K. Charypar, W. Axhausen, and K. Nagel, "An event-driven parallel queue-based microsimulation for large scale traffic scenarios," in *submitted to wctr 2007*, 2007.

[70] J. Baxter and T. Hepplewhite, "Broad agents for intelligent battlefield simulation," in *Proceedings of the 6th Computer Generated Forces and Behavioural Representation*, Institute of Simulation and Training, 1996.

[71] R. Vincent, B. Horling, T. Wagner, and V. Lesser, "Survivability simulator for multi-agent adaptive coordination," in *Proceedings of the International Conference on Web-Based Modeling and Simulation 1998 (WMC'98)*, 1998.

[72] L. Lamport, "Time, clocks and the ordering of events in a distributed system.," *Communications of the ACM*, vol. 21, no. 7, pp. 558-565, Jul. 1978.

[73] J. Misra, "Distributed discrete-event simulation," *ACM Computing Surveys*, vol. 18, no. 1, pp. 39-65, Mar. 1986.

[74] S. B. Yoginath and K. S. Perumalla, "Parallel vehicular traffic simulation using reverse computation-based optimistic execution," in *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, 2008, pp. 33-42.

[75] M. Marzolla, "Distributed simulation of large computer systems," in *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP'01)*, Beijing, P.R. China, 2001.

[76] D. Jefferson and H. Sowizral, "Fast concurrent simulation using the Time Warp mechanism,' in *Proceedings of the SCS Distributed Simulation Conference*, 1985, pp. 63-69.

[77] R. Fujimoto, "Parallel discrete event simulation,' *Communications of the ACM*, vol. 33, no. 10, pp. 30-53, 1990.

[78] K. M. Chandy and J. Misra, "Asynchronous distributed simulation via a sequence of parallel computations," *Communications of the ACM*, vol. 24, no. 11, pp. 198-205, Nov. 1981.

[79] G. Theodoropoulos and B. Logan, "A framework for the distributed simulation of agent-based systems," in *Modelling and Simulation: a tool for the next milenium, Proceedings of the 13th European Simulation Multiconference (ESM'99)*, 1999, pp. 58-65.

[80] G. Theodoropolous and B. Logan, "The distributed simulation of agent-based systems," in *Developments in Computational Mechanics with High Performance Computing: Proceedings of the Third Euro-conference on Parallel and Distributed Computing for Computational Mechanics*, 1999, pp. 147-154.

[81] B. Logan and G. Theodoropoulos, "Dynamic interest management in the distributed simulation of agent-based systems," in *Proceedings of the Tenth Conference on AI, Simulation and Planning, AIS-2000*, 2000, pp. 45-50.

[82] G. Theodoropoulos and B. Logan, "A unified framework for interest management and dynamic load balancing in distributed simulation," in *Proceedings of the Twelfth European Simulation Symposium (ESS-2000)*, 2000, pp. 11-115.

[83] B. Logan and G. Theodoropoulos, "The distributed simulation of multiagent systems," in *Proceedings of the IEEE*, vol. 89, 2001, pp. 174-185.

[84] A. Ferscha and S. K. Tripathi, "Parallel and distributed simulation of discrete event systems," University of Maryland, Technical Report cs.tr.3336, 1994.

[85] P. Fernandes and U. Nunes, "Multi-agent architecture for simulation of traffic with communications," in *ICINCO 2008- International Conference on Informatics in Control, Automation & Robotics*, Funchal, 2008.

[86] S. Y. Wang, et al., "NCTUNS 4.0: an integrated simulation platform for vehicular traffic, communication, and network researches," in *1st IEEE WiVec 2007 (International Symposium on Wireless Vehicular Communications)*, Baltimore, MD, 2007.

[87] J. Dijkstra, A. J. Jessurun, B. De-Vries, and H. J. Timmermans, "Agent Architecture for Simulating Pedestrians in the Built Environment," in *Proc. of the Fourth International Workshop on Agents in Traffic and Transportation (ATT2006)*, New York,NY, 2006.

[88] J. Dijkstra, H. J. P. Timmermans, and A. J. Jessurun, "A multi-agent cellular automata system for visualising simulated pedestrian activity," in *Theoretical and Practical Issues on Cellular Automata Proceedings on the 4th International Conference on Cellular Automata for Research and Industry*, Germany, 2000, pp. 29-36.

[89] L. Correia and T. Wehrle, "Beyond cellular automata, towards more realistic traffic simulators," in *Proc. of Cellular Automata - 7th. International Conference on Cellular Automata for Research and Industry ACRI 2006*, vol. 4173, Perpignan, France, 2006, pp. 690-693.

[90] A. Landsdowne, "Traffic simulation using agent-based modelling," BSc. Computer Science Thesis, University of the West of England, 2006.

[91] M. Chabrol, D. Sarramia, and N. Tchernev, "Urban Traffic Systems Modelling Methodology," *International Journal of Production Economics*, vol. 99, no. 1-2, pp. 156-176, Jan. 2006.

[92] M. Darbari, B. Karn, P K. Bansal, and R. Goel, "Information design for urban traffic simulation," *Georgian Electronic Scientific Journal: Computer Science and Telecommunications*, vol. 3, no. 17, pp. 108-115, 2008.

[93] R. Nicola and M. Lloreti, "A modal logic for mobile agents," *ACM Transactions on Computational Logic*, vol. 5, pp. 79-128, Jan. 2004.

[94] H. Aldewereld, W. Hoek, and J. J. Meyer, "Rational teams: logical aspects of multi-agent systems,' *Fundamenta Informaticae*, vol. 63, no. 2-3, pp. 159-183, 2004.

[95] (2008, Feb.) Agent UML. [Online]. www.auml.org/auml

[96] R. Valk, "Petri nets as token objects. An introduction to elementary object nets," in *International Conference on application and Theory of Petri nets*, 1998, pp. 1-

25.

[97] K. Jensen, *Coloured Petri nets, Volume 1: basic concepts*. Springer-Verlag, 1992.

[98] H. I. Almeyda Canepa, "Un sistema de red a tres niveles para el modelado de agentes móviles," 2002.

[99] K. Hiraishi, "A Petri-net-based model for the mathematical analysis of multi-agent systems. ,' in *Proceedings of the IEEE International Conference on Systems Man & Cybernetics*, Nashville, Tennessee, USA., 2000.

[100] E. López, N. Villanueva, and H. Almeyda, "Modelling of batch production systems using Petri nets with dynamic tokens," *Mathematics and Computers in Simulation*, vol. 67, no. 6, pp. 541-558, Jan. 2005.

[101] R. Sanchez Herrera, "Especificación Multinivel de Protocolos de Interacción en Sistemas de Agentes Móviles," 2004.

[102] H. V. D. Parunak, S. Brueckner, M. Fleischer, and J. Odell, "A design taxonomy of multi-agent interactions," in *Agent-Oriented Software Engineering IV*. Springer Berlin / Heidelberg, 2004, pp. 123-137.

[103] B. Zeigler, T. Kim, and H. Praehofer, *Theory of Modeling and Simulation*. Academic Press., 2000.

[104] T. Nagatani, "The physics of traffic jams," Institute of Physics Publishing, UK, Reports on Progress in Physics, 2002.

[105] A. W. Roscoe, *The Theory and Practice of Concurrency*. Prentice Hall, 1997.

[106] F. Baude, et al., "Grid Computing: Software Environments and Tools," in *Programming , Deploying, Composing, for the Grid*, J. C. Cunha and O. F. Rana, Eds. Springer-Verlag, 2006.

Apéndice A: Definición de n-LNS

El formalismo n-LNS está basado en la idea de redes dentro de redes. La especificación de n-LNS requiere una jerarquía entre las diferentes redes que conforman el sistema a modelar, restringiendo el marcado a símbolos o redes que tengan una posición jerárquica inferior a ella; en las transiciones hay etiquetas que cuentan con atributos de sincronización: local, interna y externa; para la sincronización local existe un valor de cardinalidad para poder especificar cuando se requiere la interacción entre más de dos redes (o agentes) en un lugar (o parte del ambiente) y en los arcos se especifica el peso respecto a cada etiqueta en la transición asociada al arco.

A.1 Especificación formal

En la tesis de R. Sánchez **[101]** está la definición formal de n-LNS la cual se incluye en este apartado para referencias posteriores que serán necesarias.

Definición A.1. Una red tipo (k de nivel i) es una tupla $typenet_{i,k} = (G, TOKEN_{i,k}, LABEL_{i,k}, VAR_{i,k}, \tau, \lambda, \chi, \pi)$ para $1 \leq i \leq n$, donde:

G es una estructura de RP ordinaria. $G = (P, T, F)$ donde :

- P es un conjunto finito no vacío de lugares
- T es un conjunto finito no vacío de transiciones
- F es una relación de flujo $P \times T \cup T \times P$, tal que $P \cap T = \varnothing$

$TOKEN_{i,k}$ es un conjunto finito no vacío de redes tipo y símbolos permitidos dentro de los lugares de una red tipo k de nivel i:

$TOKEN_{i,k} \subseteq \{typenet_{j,k} \mid i < j \leq n,\ 1 \leq k \leq r\} \cup SYMBOLS$ donde

- n es el número de niveles de un sistema de red multi-nivel
- r es el número de redes tipo diferentes permitidas dentro de los lugares de una red tipo de nivel i
- $SYMBOLS$ es un conjunto finito de símbolos.

Así, $TOKEN_{1,k} \subseteq \{typenet_{2,1}, ..., typenet_{n,k}\} \cup SYMBOLS$

$TOKEN_{2,k} \subseteq \{typenet_{3,1}, ..., typenet_{n,k}\} \cup SYMBOLS$

$TOKEN_{n,k} \subseteq SYMBOLS$

$LABEL_{i,k}$ es un conjunto finito de etiquetas definidas para la red tipo k de nivel i.

$\tau : P \rightarrow 2^{TOKENi,k}$ $\varnothing$ es una función de asignación de redes tipo y símbolos a los lugares.

$VAR_{i,k} = \{x, y, \ldots\}$ es un conjunto finito de variables definidas para la red tipo k de nivel i.

$Type$: $VAR_{i,k} \rightarrow 2^{(TOKENi,k\ -\ SYMBOLS)}$ es una función de asignación de tipos a las variables

$Type(x) = \{typenet_{i,k} \mid typenet_{i,k} \in \tau(p), p \in P_{i,k}\}$ es el conjunto de tipos asociados a la variable x.

λ: $T \rightarrow 2^{LABi,k}$ $\varnothing$ es una función de asignación de etiquetas a las transiciones, donde:

$LAB_{i,k} = (LABEL_{i,k} \times ATTRIB_i)$ y

- Si $i = 1$, entonces $ATTRIB_i = 2^{\{\downarrow\}}$
- Si $2 \leq i \leq n\text{-}1$ entonces $ATTRIB_i = 2^{\{\equiv, \downarrow, \uparrow\}}$
- Si $i = n$ entonces $ATTRIB_i = 2^{\{\equiv, \uparrow\}}$

Attrib: $LAB_{i,k} \rightarrow ATTRIB_i$ es una función que regresa el conjunto de atributos de sincronización de una etiqueta *label*.

χ: $\{(t, lab) | t \in T \wedge lab \in \lambda(t) \wedge \equiv \in Attrib(lab)\} \rightarrow N \times (N \cup \omega)\}$es una función que asigna a cada transición respecto a una etiqueta el número de transiciones (expresado como un intervalo) con las que debe sincronizarse de manera local.

π: $F_{i,k} \times LAB_{i,k} \rightarrow M_{VARi,k \cup SYMBOLS \cup CONST}$ es una función de asignación de pesos para cada arco respecto a las etiquetas de una transición; el peso es un multi-conjunto de variables y/o símbolos y constantes de red. Si $label \notin \lambda(t)$, entonces $\pi((p, t), label) = \pi((t, p), label) = \varnothing$. Además, si $i = n$ entonces $VAR_{i,k} = \varnothing$, y π: $F_{n,k} \times LABEL_{n,k} \rightarrow M_{SYMBOLS}$. *CONST* es un conjunto de redes de nivel inferior que pueden estar contenidas en P. Estas redes serán definidas más adelante.

Una red tipo $typenet_{i,k}$ es una estructura de RP ordinaria con información adicional que declara y manipula los datos definidos en $TOKEN_{i,k}$ de acuerdo a las Pre- y Post- condiciones establecidas por la función π, y al etiquetado simbólico de transiciones especificado por λ, para la interacción entre redes.

De acuerdo a la definición, las redes están organizadas en niveles: un nivel i puede contener una o más redes de diferente tipo. Cada red de tipo k y nivel i puede contener tipos definidos en el conjunto $TOKEN_{i,k}$. La función τ asigna un subconjunto de $TOKEN_{i,k}$ a cada lugar de G, indicando que el lugar puede contener como marcas únicamente elementos de dicho subconjunto (símbolos y/o redes tipo).

La función λ asigna a cada transición un conjunto de tuplas de la forma (*label*, *attrib*), donde *label* es un elemento de $LABEL_{i,k}$, y *attrib* es un subconjunto de $ATTRIB_i$, que representa los tipos de sincronización asociados a la etiqueta *label*. Por ejemplo, la tupla $(l,\varnothing)$ indica que la etiqueta l no requiere sincronización alguna (disparo autónomo), mientras que $(l,\{\uparrow\})$ denota que l requiere sincronización *externa*; $(l,\{\downarrow\})$ es para sincronización *interna*, y $(l,\{\equiv\})$ indica sincronización local. Los demás subconjuntos de *ATTRIB* representan combinaciones de tipos de sincronización. Así, la tupla $(l,\{\equiv, \downarrow,\uparrow\})$ indica que l requiere sincronización *local*, *interna* y *externa*. Por simplicidad, escribiremos de aquí en adelante $l^{\equiv\downarrow\uparrow}$ en lugar de utilizar la notación anterior, y cuando *attrib* es vacío, escribimos solo l.

Los elementos del conjunto $ATTRIB_i$ varían de acuerdo al nivel al que pertenece G. Si la red es de nivel 1, entonces sus transiciones sólo pueden sincronizarse internamente, por lo que $ATTRIB_i$ es igual al conjunto potencia de $\{\downarrow\}$; si la red pertenece al nivel n, entonces está permitido la sincronización local y externa de sus transiciones, por lo que $ATTRIB_i$ es igual al conjunto potencia de $\{\uparrow,\equiv\}$. Si la red es de un nivel intermedio (mayor que 1 y menor que n), entonces se permite los tres tipos de sincronizaciones, y $ATTRIB_i$ es el conjunto potencia de $\{\downarrow,\uparrow,\equiv\}$.

La función χ determina la cardinalidad de una transición t, esto es, el número de redes con las que t se debe sincronizar de manera local. χ asigna a cada transición, respecto a una etiqueta que tiene que sincronizar localmente, una pareja de enteros positivos; el primero representa el número mínimo de transiciones, y el segundo el máximo. El símbolo ω representa un número muy grande (infinito). Utilizaremos la siguiente notación: escribiremos [x, y] para representar la pareja (x, y) y [x] para representar a (x, x); por convención, si en el modelo no se indica lo contrario, $\chi(t, lab) = (1, \omega)$ para toda transición t que deba sincronizarse localmente.

La función π establece las precondiciones y las post-condiciones de cada transición de la red. Asigna un peso a cada arco con respecto a una etiqueta. Determina la cantidad y el tipo de redes y/o símbolos que se necesitan dentro de los lugares de entrada para habilitar una transición, y la cantidad y el tipo de redes y/o símbolos que deben ser agregados a los lugares de salida. Cuando la red tipo es de nivel *n*, la función π sólo asigna un multi-conjunto de símbolos al peso de los arcos, por lo tanto, la red tipo es similar a una RPC.

Una red tipo no tiene restricción alguna sobre el peso de los arcos. Es posible especificar en el peso de los arcos de salida de una transición, símbolos y/o variables del tipo nc incluido en los lugares de entrada a dicha transición, e inclusive omitir uno o varios de ellos. De esta manera, una transición puede:

- Eliminar redes y/o símbolos, como resultado de consumirlos y no agregarlos a un lugar de salida.
- Crear redes y/o símbolos en un lugar de salida cuando no han sido removidos de sus lugares de entrada , como resultado del disparo de la transición.
- Clonar redes y/o símbolos que se obtienen de removerlos y sumar varias copias de estos.

A.2 Redes de nivel *i*

Las redes de nivel *i* están formadas por una red tipo y un marcado inicial.

Definición A.2. Una red *k* de nivel *i* es una tupla $NET_{i,k} = (typenet_{i,k}, \mu_{i,k}) \mid 1 \leq i \leq n, k = 1, \ldots r$, donde:

$typenet_{i,k}$ es una red tipo *k* de nivel *i*.

$\mu_{i,k} : P_{i,k} \rightarrow M_{NETS_{TOKENi,k}} \cup \text{SYMBOLS}$ es una función de marcado para la red *k* de nivel *i*.

$NETS_{TOKEN_{i,k}} \subseteq \{NET_{i+1,k}, NET_{i+2,k}, \quad , NET_{n,k}\}$ es el conjunto de todas o algunas redes de nivel inferior que están como marcas de la red $NET_{i,k}$.

La función de marcado $\mu_{i,k}$ le asigna a cada lugar de G un multi-conjunto de redes y de símbolos, cuyas redes tipos y símbolos se encuentran en el conjunto de tipos y símbolos asignados al lugar por la función τ.

El tipo de una red se determina mediante la función $TypeN(NET_{i,k}) = typenet_{i,k}$.

A.3 Sistema n-LNS

Un sistema n-LNS, o un sistema de red, es una colección de una o más redes instanciadas a partir de cada una de las redes tipo definidas en todos los niveles.

Definición A.3. Un sistema de red a *n* niveles es una tupla $n-LNS = (NET_{i,k} \mid \exists i = 1..n : n \in N, k = 1,.., r)$, donde:

$NET_{1,1}$ es una red de nivel 1. Representa el nivel más alto del sistema de red.
$NET_{i,k} = \{NET_{2,1}, \ldots NET_{i,r}\}$ es el conjunto de *r* redes de nivel i. Representa las redes de nivel intermedio y nivel más bajo.

A.4 Evolución del marcado

Mientras que la parte estática de una red está representada por su estructura, la parte dinámica está caracterizada por la evolución de su marcado, mediante las reglas de habilitación y disparo de sus transiciones.

Antes de definir estas reglas, debemos introducir algunos conceptos.

Definición A.4. Una función de ligado b sobre un conjunto de variables $VARS$ = {x, y, ...} es una función b: $VARS \rightarrow NETS_{TOKEN_{i,k}}$; para una variable $v \in VARS$, b(v) es una red de nivel inferior cuyo tipo pertenece a $Type(v)$. $m<b>$ denota el multi-conjunto de redes que resulta de evaluar un multi-conjunto de variables m en un ligado b.

Definición A.5. $b_t : \left\{ v \mid v \in \bigcup_{p \in \bullet t} VAR(E_{\pi((p,t),label)}) \right\} \rightarrow \bigcup_{p \in \bullet t} NETS_{TOKEN_i}(p)$ es una función que asigna a cada variable definida en el peso de los arcos de entrada a la transición t, con respecto a la etiqueta $label$, redes del tipo asignado al conjunto de lugares de entrada a t, donde $NETS_{TOKEN_i}(p) = \{net \mid net \in NETS_{TOKEN_i} \wedge TypeN(net) \in \tau(p)\}$, $E_{\pi((p,t),label)}$ denota los elementos sin repetición del multi-conjunto $\pi((p, t), label)$ y VAR($E_{\pi((p,t),label)}$) el conjunto de variables en $E_{\pi((p,t),label)}$.

A.5 Regla de habilitación

Definición A.6. Una transición t de una red k de nivel i $NET_{i,k}$ está habilitada con respecto a una etiqueta $lab \in \lambda(t)$ si se verifican las siguientes condiciones:

1. Existe un ligado b_t si $VAR_t \neq \varnothing$; si $VAR_t = \varnothing$, como en el caso de las redes de nivel n o cuando en el peso de los arcos incluye sólo símbolos y constantes de red, entonces el ligado es omitido; VAR_t es el conjunto de variables que aparecen en todos los arcos de entrada a t.
2. $\forall p \in \bullet t$, $\pi((p, t), lab) <b_t> \subseteq \mu_{i,k}(p)$ si $VAR_t \neq \varnothing$, ó $\forall p \in \bullet t$, $\pi((p, t), lab) \subseteq \mu_{i,k}(p)$ si $VAR_t = \varnothing$; en cada lugar de entrada a t hay tantas redes de nivel inferior y símbolos como lo especifica la función π evaluada en el ligado b_t. El ligado $<b_t>$ es omitido cuando el peso de los arcos no contiene variables.
3. Las condiciones de uno de los siguientes casos se cumplen:

   Caso 1: Si $lab = (l, \varnothing)$, el disparo de t no requiere sincronización, se realiza de manera autónoma.

   Caso 2: Si $lab \neq (l, \varnothing)$ se debe considerar una o una combinación de las siguientes situaciones:

   i) $lab = (l, \{\equiv\})$ (Sincronización local). Se requiere la habilitación simultánea de las transiciones etiquetadas con $l^{\equiv}$ pertenecientes a otras redes que se encuentran en el mismo lugar p' de una red de nivel superior. Además, si $\chi(t, lab) = (x, y)$, el número de éstas redes debe ser mayor o igual a x y menor o igual a y. El disparo de estas transiciones es simultáneo, y todas las redes sincronizadas permanecen en p'

ii) $lab = (l, \{\downarrow\})$ (Sincronización interna). Se requiere la habilitación simultánea de las transiciones etiquetadas con $l^{\uparrow}$ pertenecientes a otras redes de nivel inferior que se encuentran en $\bullet t$. Estas transiciones se disparan simultáneamente y las redes de nivel inferior y símbolos especificados en $\pi((p, t), lab)\langle b_t \rangle$ son removidos.

iii) $lab = (l, \{\uparrow\})$ (Sincronización externa). Se requiere la habilitación de al menos una de las transiciones $t' \in p'\bullet$ etiquetadas con $l^{\downarrow}$ de la red de nivel superior donde está contenida la red $NET_{i,k}$. El disparo de t provoca la transferencia de $NET_{i,k}$ y los símbolos declarados en $\pi((p', t'), lab)\langle b_t \rangle$.

Una etiqueta puede tener más de un tipo de sincronización. Por ejemplo, una etiqueta $lab = (l, \{\uparrow, \downarrow, \equiv\})$, debe sincronizarse de manera local, interna y externa respecto al símbolo l.

Es importante señalar que la sincronización interna no tiene efecto cuando una red de nivel i se sincroniza internamente a través de una transición t, respecto a una misma etiqueta, con una transición t'' de una red de nivel $i + 2$, la cual está contenida en una red de nivel $i + 1$, que se encuentra marcando $\bullet t$ de la red de nivel i, y cuyo símbolo no está asociado a la transición de salida t'. Esto mismo ocurre para el caso de sincronización externa.

A.6 Regla de disparo

La cantidad de redes y/o símbolos que una transición produce o consume al dispararse se indica en el peso de los arcos de entrada y salida a la transición.

Definición A.7. El marcado de la red que se obtiene al disparar una transición t de una red k de nivel i $NET_{i,k}$ está dado por las siguientes expresiones:

$$\forall p \in \bullet t \cup t\bullet\ ,\ \mu'_{i,k}(p) = \mu_{i,k}(p) - \pi((p, t), label) \cup \pi((t, p), label) \quad \text{si } VAR_t = \varnothing$$

$$\forall p \in \bullet t \cup t\bullet\ ,\ \mu'_{i,k}(p) = \mu_{i,k}(p) - \pi((p, t), label)\langle b_t \rangle \cup \pi((t, p), label)\langle b_t \rangle \quad \text{si } VAR_t \neq \varnothing \wedge \downarrow \notin Attrib(label).$$

$$\forall p \in \bullet t \cup t\bullet\ ,\ \mu'_{i,k}(p) = \mu_{i,k}(p) - \pi((p, t), label)\langle b_t \rangle \cup \pi((t, p), label)\langle b_{t\rightarrow} \rangle \quad \text{si } VAR_t \neq \varnothing \wedge \downarrow \in Attrib(label).$$

donde:

$\mu'_{i,k}(p)$ es el marcado obtenido después de disparar la transición t.
$\mu_{i,k}(p)$ es el marcado actual antes de disparar t.
b_t es el ligado que habilita a t.
$b_{t\rightarrow}$ es un ligado con el dominio igual a b_t y cuyo rango son las redes en el rango de b_t pero con su marcado alcanzado al disparar sus transiciones que deben sincronizarse con t.
VAR_t es el conjunto de variables que aparecen en todos los arcos de entrada a t.

Si t requiere algún tipo de sincronización, deben dispararse de manera simultánea junto con t todas las transiciones que estén habilitadas respecto a *label*.

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

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

Simulación Distribuida de Tráfico Urbano Basada en Sistemas Multi-Agentes

del (la) C.

Emmanuel LÓPEZ NERI

el día 16 de Octubre de 2009.

Dr. Juan Manuel Ramírez Arredondo
Investigador CINVESTAV 3C
CINVESTAV Unidad Guadalajara

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

Dr. Victor Manuel Larios Rosillo
Coordinador del DOctorado en
Tecnologías de la Información
Universidad de Guadalajara