UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 5

Detección de infracciones del conductor en un simulador de auto

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores: Luis Felipe Lores Caignet

Yunior Michel Marty Consuegra

Tutor: Lic. Yuniesky Coca Bergolla

Ciudad de la Habana

Mayo, 2007

Declaración de autoría

Por este medio declaramos que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas (UCI) para que haga el uso que estime pertinente con este trabajo.

Para que así conste firmamos la presente a los _____días del mes de _____ del _____.

_____ _____

Firma del Autor
(Luis Felipe Lores Caignet)

Firma del Autor
(Yunior Michel Marty Consuegra)

_____

Firma del Tutor
(Lic. Yuniesky Coca Bergolla)

Datos de Contacto

Lic. Yuniesky Coca Bergolla
e-mail: ycoca@uci.cu
Telf.: 8372474

Graduado de Ciencias de la Computación en la Universidad Central de Las Villas en el año 2003. Profesor de la Universidad de las Ciencias Informáticas desde ese mismo año. Es Jefe de Dpto. de las asignaturas de la especialidad de la Facultad 5 “Entornos Virtuales”, además de ser líder del Grupo de Desarrollo de Elementos Virtuales Inteligentes. Obtuvo el Sello Forjadores del Futuro en el 2007.

Agradecimientos

… a la Universidad de las Ciencias Informáticas que nunca olvidaremos por esta gran oportunidad que nos ha brindado.

… a la Empresa SIMPRO y a todo su personal que nos ha dado la oportunidad de desarrollar esta tesis.

… al Licenciado Yuniesky Coca Bergolla por su gran apoyo como tutor y ejemplo brindado en cada instante.

Dedicatoria

… a mis padres Julio y Regina que sin ellos nunca hubiese realizado este gran sueño y culminar mis estudios con éxito.

… a mi hermano Julio Cesar y amigos por darme su ejemplo.

… a mi sobrina Lorena para que le sirva como guía y ejemplo a seguir.

Luis F.

Resumen

Las aplicaciones de realidad virtual actualmente han logrado un realismo tan elevado que constituyen herramientas fundamentales para la medicina, el diseño, la formación, el entrenamiento, etc. Muchas empresas de reconocido prestigio dedican grandes recursos para el desarrollo de simuladores basados en esa tecnología. En Cuba una de las empresas más destacadas en este sector es la Empresa SIMPRO la cual ha obtenido un gran desarrollo y prestigio a nivel nacional e internacional.

La empresa SIMPRO en unión con la Universidad de las Ciencias Informáticas realizan un proyecto dedicado a los simuladores virtuales, en la actualidad se han tenido grandes resultados con la elaboración de un simulador de auto, el cual ha sido presentado en importantes eventos como la feria de transporte 2006 y 2007. En el presente, uno de los principales problemas que afecta al simulador es la detección de las diversas infracciones que pudiera cometer el conductor.

Como propuesta de solución a la problemática antes expuesta este proyecto aborda la implantación de un módulo para la detección de infracciones de tránsito y mecánicas cometidas por el conductor del simulador de conducción. Con el resultado de este módulo se dotará al Simulador de auto SIMPRO de un eficiente sistema de Detección de infracciones que permitirá que el resultado del entrenamiento virtual de conducción de los usuarios tenga una mayor efectividad en el cumplimiento de las leyes de conducción establecidas.

Palabras claves: Simulador de conducción, Infracciones de tránsito, entorno virtual de ciudad.

Contenido

Introducción.

En los últimos años los simuladores de conducción se han hecho indispensables en el aprendizaje o entrenamiento en distintas áreas: en las autoescuelas los alumnos pueden enfrentarse a situaciones críticas, por ejemplo conducir en situaciones extremas ya sea en terrenos nevados, rocosos, etc., antes incluso de saber conducir. En el caso de los pilotos de avión es evidente la necesidad de los simuladores. En resumen con los simuladores se pueden entrenar muchas de las situaciones que son imposibles de recrear en la realidad, a un coste económico inferior y sin comprometer la seguridad de los conductores. Gracias a los simuladores es posible aumentar la seguridad de los operarios y usuarios así como mejorar la utilización de los vehículos por parte de estos.

En la actualidad la Realidad Virtual irrumpe en el mercado poniendo a disposición de cualquier usuario dicha tecnología aun incipiente pero con un horizonte impresionante de desarrollo, de esta necesidad de incluirse en este naciente mercado es que en nuestro país y especialmente en la Universidad de las Ciencias Informáticas se comienza a desarrollar el Simulador de auto SIMPRO en unión con la Empresa SIMPRO.

Desde su inicio en el año 2002 el simulador de auto SIMPRO ha alcanzado prestigio y aceptación dentro y fuera del país. Sin embargo está lejos de ser el simulador al que se aspira.

Al transitar por una urbe con un tráfico de autos considerables se provocan fácilmente desastres automovilísticos. Se podría deducir que el principal problema del tráfico en la actualidad al haber tanta circulación en las vías es causado fundamentalmente por la falta de precaución y de destreza al conducir. En el simulador de auto SIMPRO no se cuenta con un sistema que posibilite la prevención de infracciones del conductor al ingresar en su paseo virtual, todo esto ha traído consigo una falta de realismo importante en el simulador y un futuro descontento de los usuarios que obtendrán los servicios finales de este producto. La mayor importancia de la prevención de

infracciones en un simulador de conducción recae en el seguro de vida que representaría para los conductores conocer y prevenir dichas infracciones, mucho antes de enfrentarse a la práctica real de conducción.

Por tanto se hace necesario desarrollar un sistema que permita darle solución a los problemas mencionados anteriormente, con esta convicción y espíritu de superación, se propone este trabajo para brindar las ideas esenciales para la incorporación de la necesaria detección de infracciones del conductor en el simulador, que perfectamente pudiera ser incrementada de manera sencilla y generalizada para otros simuladores en entornos urbanos fundamentalmente.

El objeto de estudio de la presente tesis son los simuladores virtuales, delimitando así el campo de acción a la detección de infracciones de un conductor en el simulador de autos SIMPRO.

Con el desarrollo del Módulo de detección de infracciones de conducción en el simulador de auto SIMPRO y para alcanzar el realismo y eficiencia antes mencionado, el objetivo general es desarrollar un módulo que permita la detección y clasificación de las infracciones cometidas por el conductor en el simulador de auto SIMPRO.

Se tiene como objetivos específicos:

- Detectar las infracciones de tránsito cometidas por el conductor en el simulador de auto SIMPRO.
- Clasificar las infracciones de tránsito cometidas por el conductor en el simulador de auto SIMPRO.
- Realizar una propuesta de solución para una futura implementación de las infracciones mecánicas cometidas por el conductor en el simulador de auto SIMPRO.
- Integrar a la Herramienta del Simulador de auto SIMPRO el Módulo de detección de infracciones de conducción.

Para el cumplimiento de los objetivos planteados anteriormente se definieron las siguientes tareas:

- Realizar un estudio del problema y la situación actual.
- Seleccionar la metodología a utilizar.
- Seleccionar las herramientas a utilizar para el desarrollo de la aplicación.
- Realizar un análisis y diseño del sistema, utilizando RUP como metodología.
- Capturar los requisitos funcionales y no funcionales del sistema.
- Implementar el Módulo de detección de infracciones del conductor en el simulador de auto SIMPRO.

El contenido de este trabajo está desglosado en 4 capítulos:

- El Capítulo I. Fundamentación teórica: Recoge los conceptos y características que se necesitan dominar, como simuladores virtuales y simuladores virtuales de conducción, etc., se realizará un estudio del estado del arte.
- El Capítulo II. Módulo Infracciones del conductor: Se realizará un análisis de los distintos tipos de infracciones así como los métodos que se utilizarán para resolver la problemática a la que se enfrenta el presente trabajo.
- El Capítulo III. Características del sistema: Se expone el objeto de estudio, la situación problémica y el problema, se describen las herramientas a utilizar y la metodología de desarrollo utilizada, se plantea el modelo del dominio, la captura de requisitos funcionales y no funcionales, así como la descripción de los casos de uso del sistema.
- El Capítulo IV. Análisis y diseño del sistema: Se exponen los diagramas de clases, de secuencia, de despliegue, por último un diagrama de componentes con la descripción de sus respectivos componentes.

Capítulo 1 : Fundamentación Teórica.

1.1 Introducción.

En el presente capítulo se realizará un estudio de los conceptos y características principales que se necesitan dominar para tener un conocimiento previo de la problemática a la que se enfrenta el presente trabajo, lo que posibilitará conocer todo lo referente a la base teórica que fundamenta la presente investigación, donde se realiza un estudio del estado del arte.

1.2 Introducción a los simuladores.

Es posible definir la simulación como: “Un intento de no cubrir, por medio de desarrollos matemáticos importantes, propiedades de un fenómeno físico, mediante la construcción de procesos análogos que reproduzcan las principales propiedades”. [Cabanellas, 2005]

Es decir, utilizar modelos simplificados de un proceso o sistema real con el objetivo de reproducir de manera suficiente aquellos comportamientos del proceso o sistema que interesa estudiar, pero eliminando aquellos otros que no se consideran representativos y además complican el desarrollo. [Cabanellas, 2005]

La utilización de la capacidad actual de los ordenadores para reproducir de manera eficiente un entorno real, permite, la realización de modelos matemáticos complejos y su resolución en un tiempo adecuado, para extraer conclusiones del comportamiento o reproducir dicho comportamiento con objetivos de formación.

En concreto, el objetivo de todo simulador será reproducir cada una de las partes del sistema o proceso de la forma más realista posible, pero siempre manteniendo bajo control la complejidad del modelado. Esta variable es fundamental pues podría hacer crecer de forma desmesurada el desarrollo sin aportar grandes mejoras. Por tanto, es necesario alcanzar un compromiso entre la profundidad del modelado y las funcionalidades que se desean retener de los sistemas reales, de este compromiso entre profundidad del modelado y complejidad depende en gran parte el éxito del Simulador.

El incremento casi exponencial de la potencia del Hardware, unido a la bajada de precio del mismo como al desarrollo a gran escala de software hace que la accesibilidad a simuladores haya llegado a un nivel casi doméstico, entiéndanse las consolas de juegos actuales como simuladores de alcance y propósitos más restringidos y para su empleo en el ocio.

Los campos de aplicación de los simuladores son cada vez más. El ocio está ganando un desarrollo notable en este mundo y aplicaciones para la educación, la rehabilitación y entrenamiento de discapacitados son ya frecuentes.

1.2.1Tipos de simuladores.

Los simuladores pueden ser clasificados atendiendo a su objetivo en los siguientes tipos: [Cabanellas, 2005]

- Diseño.
- Formación.
- Pruebas.
- Recreación.
- Operación.
- Mantenimiento.

Diseño.

Utilización de los simuladores para la reproducción de un proceso o sistema con el objetivo de realizar variaciones en los parámetros característicos del modelo y obtener conclusiones sobre el comportamiento del proceso o sistema real. Con esas conclusiones se pretende optimizar el funcionamiento del proceso.

No presentan requerimientos de tiempo real y la interfaz no necesita reproducir ningún elemento de la realidad, sin embargo los modelos matemáticos deben tener una complejidad y precisión elevadas.

Formación.

Este tipo de simuladores tienen por objeto reproducir un sistema de forma que la interacción con el usuario sea lo más realista posible, de esta forma son utilizados para entrenar a los usuarios en su relación con el sistema.

El éxito de la simulación para entrenamiento radica en no sólo entrenar la parte consciente del usuario sino “engañar” o hacer creer al subconsciente del entrenamiento.

El diseño de un simulador para formación deberá garantizar las siguientes ventajas:

- Reducción significativa del tiempo de formación.
- Entrenamientos intensivos en situaciones críticas o de emergencia, hasta su completo dominio.
- Reducción del estrés de formación en máquina real.
- Posibilidad de adaptación de los niveles de formación dependiendo de cada persona.
- Entrenamiento en aspectos generales independientemente de los diferentes tipos de trenes o tranvías.
- Posibilidad de evaluación conjunta con el alumno una vez finalizado el ejercicio.

Presentan requerimientos de tiempo real, los modelos matemáticos requeridos deben ser bastante completos para reproducir fielmente los eventos mostrados al usuario, y la interfaz necesita reproducir de forma precisa aquella del elemento o sistema que representa.

Pruebas.

En este tipo de simuladores se trata de reproducir el entorno que rodea a un determinado elemento o sistema, simulando las mismas entradas que este recibiría cuando se encontrase funcionando en el entorno real. Con estas condiciones se somete al elemento o sistema a una serie de pruebas que permitan validar el comportamiento del mismo, con la gran ventaja de realizarlas en un entorno controlado y libre de riesgos.

De esta forma se consigue comprobar la funcionalidad del elemento o sistema antes de instalarlo en su emplazamiento final, sometiéndolo a una batería de pruebas amplia y de bajo coste.

Presentan requerimientos de tiempo real, los modelos matemáticos requeridos deben ser muy completos para reproducir fielmente las entradas al equipo, y la interfaz no necesita reproducir aquella del elemento o sistema que representa.

Operación.

Los simuladores con objetivo de operación reproducen un sistema o elemento en su comportamiento de cara al usuario, incluyendo además las interfaces del mismo con el resto del sistema. De esta forma, es posible realizar acciones sobre el elemento, siendo estas interpretadas por el sistema de la misma forma que lo haría si se utilizase el elemento real.

Presentan requerimientos de tiempo real y la interfaz no necesita, aunque habitualmente es de gran utilidad, reproducir el aspecto del elemento o sistema que representa.

Mantenimiento.

Este tipo de simuladores son diseñados para proporcionar una ayuda en el mantenimiento de un determinado elemento o sistema. Se caracterizan por incluir no solo el modelado del elemento, sino también información adicional del mismo como manuales de usuario, plan de mantenimiento, niveles de stock en almacenes, etc. También pueden aportar información adicional del elemento que permita planificar el mantenimiento de una forma más sencilla, por ejemplo dimensiones de las áreas a utilizar a fin de conocer las dimensiones máximas del equipo que se puede emplear: camiones, grúas, herramientas.

Este tipo de simuladores constituyen una herramienta integrada de información, bajo un interfaz de usuario totalmente amigable al poder incorporar componentes de realidad virtual.

No presentan requerimientos de tiempo real, la interfaz debe al menos en parte reproducir el sistema real.

Recreación o reconstrucción.

El objetivo de estos simuladores es intentar reproducir un evento complejo que se ha producido: reconstrucción de accidentes de tráfico.

En general deben incorporar unos modelos matemáticos muy completos al ser los fenómenos que se pretende reproducir de una complejidad elevada.

No presentan requerimientos de tiempo real y la interfaz no necesita reproducir ningún elemento de la realidad pero es conveniente en algunos casos para hacerse una idea más intuitiva del resultado de la simulación.

Ocio y entretenimiento.

Actualmente en un pujante auge estos simuladores tienen el objetivo de entretener o hacer pasar emociones intensas a un amplio espectro de personas. Lo más importante es que la imaginación y sugestión funcionen a pleno rendimiento en el usuario, no importando tanto la fidelidad a la realidad.

Dado que el usuario no tendrá, en una proporción muy elevada, formación ni interés en los fenómenos físicos subyacente, ni tampoco hay finalidades formativas, los modelos matemáticos que se emplean son sencillos pero rápidos, y lo más complejo es afinarlos para que produzcan el grado de emociones subjetivas que se espera de ellos.

El requerimiento de tiempo real es obvio y la interfaz visual debe ser naturalmente lo más realista posible, dado que es por la vista por donde se capta la mayoría de la información del entorno exterior. El acompañamiento de movimiento, vibraciones y sonido mejora sustancialmente el grado de emociones que este tipo de simuladores presenta.

1.2.2Reseña Histórica.

Los orígenes de los simuladores virtuales se remontan a la segunda mitad del siglo pasado pero ya con anterioridad se habían concretado algunos proyectos en los cuales se evidenciaba la terminología de simulación, la mayoría para no ser absolutos eran referentes a la aviación:

- En 1941 simulador "Celestial Navigation Trainer": fue una enorme estructura de 13,7 metros capaz de acomodar en su interior una tripulación entera de un bombardero para aprender como volar en misiones nocturnas.
- En 1948, Curtiss-Wright desarrolló un simulador para el Stratocruiser de Pan American, el primer simulador de vuelo completo utilizado por una aerolínea. Aunque no se había

simulado movimiento ni vistas exteriores. A partir de ese momento los sistemas de movimiento llegaron y se instalaron a finales de los años 50.

- En 1954 General Precision Inc. desarrolló un simulador con movimiento el cual contenía una cabina dentro de un marco metálico.
- En los años 60's es cuando emergen los primeros simuladores de vuelo construidos para la fuerza aérea estadounidense, donde los estudiantes de piloto aprendían a manejar aviones entrenando en cabinas aéreas montadas en plataformas movibles las cuales subían, bajaban o movían hacia los lados dependiendo de las acciones que el piloto realizara sobre los controles.
- Otro importante concepto fue el "Sensorama" una máquina creada por Morton Heiling en 1962, esta máquina simulaba las experiencias sensoriales de un paseo en motocicleta, al combinar imágenes, sonido, viento y aromas. En esta experiencia el usuario subía en el asiento de una motocicleta, tomaba los manubrios y usaba un visor parecido a unos binoculares donde podía pasear por dunas en California o las calles de Brookly, y donde unos pequeños 'grills' cerca de la nariz del usuario emitían aromas auténticos. Sin embargo, este proyecto era sumamente complejo y nunca se logró materializar versiones más sencillas de éste.
- En 1972 es cuando se desarrolla el primer simulador de vuelo computarizado, el cual fue importante para el despegue de la realidad virtual.
- En 1989 el Departamento de defensa de los EE.UU. crea a Simnet (Simulador de Red), una red experimental de estaciones de trabajo basadas en microprocesadores que habilitaban al personal a prácticas de operaciones de combate interactivamente en sistema de entrenamiento de tiempo real. De hecho este sistema se utilizó para entrenar al ejército en la Guerra del Golfo Pérsico.

1.3 Simuladores de Conducción.

El desarrollo de simuladores es un tema que va tomando cada vez más importancia en la sociedad en la que vivimos. Desde los simuladores para las actividades espaciales y militares se está llegando a una situación en que la mayoría de las industrias y empresas de transporte están empleando simuladores para distintas fases de su actividad, desde diseño y planificación hasta formación de conductores y operarios.

Los sistemas de arranque activados ahora simulan los ruidos y las vibraciones de un vehículo real, ya sea con transmisión manual o automática. El simulador permite a los conductores sentarse al volante, pasar los cambios e interactuar con los instrumentos medidores. [Nikkel, 2002]

Un simulador de conducción pretende reproducir tanto las sensaciones físicas como la velocidad, la aceleración, la percepción del entorno, etc. como el comportamiento de los equipos de la máquina que se pretende simular. Para simular las sensaciones físicas se puede recurrir a complejos mecanismos hidráulicos comandados por potentes ordenadores que mediante modelos matemáticos consiguen reproducir sensaciones de velocidad y aceleración.

En un simulador de conducción, el conductor entra al simulador o cabina, y se enfrenta a una proyección computadorizada que muestra escenarios virtuales en plena conducción. Para la persona en la cabina, la ilusión es muy completa, y totalmente real, y piensan que realmente están conduciendo ya sea un avión, auto, etc. En este sentido, es posible trabajar con procedimientos de emergencia, y con situaciones extraordinarias para lograr un realismo en el cual el conductor sienta la sensación de que realmente está conduciendo y se está enfrentando a la situación real que él cotidianamente se enfrenta al conducir, sin poner en peligro su vida.

Los Simuladores virtuales de conducción entran en el área de la Realidad Virtual de tipo sistemas no inmersivos o de escritorios, lo que ha traído consigo un gran desarrollo de estos en los últimos años, propiciado por el descenso de los precios de las herramientas utilizadas en los Sistemas de Realidad Virtual no inmersivos en el mercado mundial, esto ha contribuido a que un

sin números de compañías de disímiles países se animaran a ingresar en este mercado aun naciente y en plena evolución.

Existen diversos tipos de simuladores que atienden a objetivos y necesidades diferentes, el presente trabajo se centrará en los simuladores de conducción, concretamente simuladores de conducción de autos y esencialmente en el control de tráfico vehicular del conductor del simulador.

El objetivo principal de un simulador para entrenamiento de conducción es que sus usuarios aprendan a conducir de forma eficiente sin tener que acceder a la práctica real. La adquisición de nuevas aptitudes requiere la exposición a un conjunto de situaciones como la conducción en condiciones adversas, o bajo mayores exigencias de rendimiento o de precisión que las de la conducción en la realidad. Por este motivo la utilización de simuladores virtuales para el aprendizaje presenta importantes inconvenientes, tales como:

- **Coste:** La dedicación de estos simuladores al aprendizaje supone un coste muy elevado.
- **Riesgo:** Ciertos ejercicios beneficiosos desde el punto de vista del aprendizaje pueden conllevar un alto riesgo, o ser irrealizables en un escenario real.

La utilización de un simulador para el entrenamiento permite además un seguimiento detallado de los aprendices por personal experto. Además, un simulador es una herramienta que, aparte de realizar labores de formación, puede ser empleado para muchos otros propósitos, entre los múltiples usos que se le pueden dar a un simulador de entrenamiento se encuentran los siguientes:

- Reducción del tiempo de formación, puesto que se emplean diseños instruccionales que favorecen la asimilación de habilidades en la secuencia adecuada.
- Repetición de un determinado ejercicio tantas veces como el instructor considere necesario sin necesidad de asumir los gastos asociados a la utilización de la máquina real.

- Entrenamiento con conductores para que sepan reaccionar ante situaciones de emergencia y situaciones excepcionales.
- Entrenamiento de los hábitos de los usuarios durante la operación, que mejoran el rendimiento y reducen las averías del automóvil.

1.4 Análisis de otras soluciones existentes.

Simulador SIMPRO.

En nuestro país, una de las empresas que ha impulsado el desarrollo de los Sistemas de Realidad Virtual, es el "Centro de Investigación y Desarrollo #2", conocido en el mercado como SIMPRO. Esta empresa comenzó como proyecto desde el año 1994, y oficialmente existe con el nombre de SIMPRO desde el año 2000, a partir del año 2002 la Universidad de las Ciencias Informáticas y la empresa SIMPRO comenzaron a trabajar en conjunto, como fruto de esta integración se han logrado un sin número de trabajos pero el que ha tenido mayor relevancia a nivel nacional e internacional ha sido el Simulador de conducción SIMPRO. La empresa SIMPRO esta avalada entre otras cosas con numerosos premios a las investigaciones y calidad de sus productos desde su fundación, entre sus numerosos premios se encuentran:

- Premio de la Academia de Ciencias a las investigaciones más destacadas del año en dos ocasiones. (1996, 1998).
- Dos premios relevantes en Fórum Nacional de Ciencia y Técnica.
- Premios en la Feria Internacional del Transporte a la calidad del producto.
- Premio de la Academia de Ciencias a las investigaciones más destacadas de año 1996.
- Premio de la Academia de Ciencias a las investigaciones más destacadas de año 1998.

En Cuba y especialmente en la empresa SIMPRO no se ha desarrollado un módulo para la detección de infracciones en un simulador de conducción que pueda servir de base para dar solución a la problemática que se pretende resolver en este trabajo, quizás por la falta de

recursos que se tenía en nuestro país hace algunos años no se pensó en realizar un proyecto tan abarcador como el Simulador de auto que hubiera sido de gran utilidad para el desarrollo del presente trabajo, por esta razón se han tenido que elaborar técnicas o métodos para la detección de las infracciones cometidas por el conductor en el simulador. En el siguiente capítulo se explican en detalle los métodos utilizados para la detección de las infracciones del conductor en el simulador.

1.5 Conclusiones.

En este capítulo se ha podido conocer la base teórica que fundamenta la presente investigación, se trataron conceptos fundamentales relacionados con la investigación, se acercó al lector al conocimiento de los simuladores virtuales y su auge en la actualidad, y se llegó a la conclusión de que independientemente de la gran variedad de simuladores virtuales que existen en la actualidad, los destinados al entrenamiento y entretenimiento son de los más difundidos en la población mundial, de aquí la necesidad de incorporar la necesaria detección de infracciones del conductor en el Simulador de auto SIMPRO, con lo cual se lograría dar cumplimiento al objetivo general de este trabajo, por último se realizó un recuento de la trayectoria de la empresa SIMPRO.

Capítulo 2 : Módulo Infracciones del conductor.

2.1 Introducción.

En el presente capítulo se realizará un análisis profundo del Módulo de detección de Infracciones del conductor en el simulador SIMPRO, lo cual permitirá conocer importantes aspectos que ayudarán a dar solución a la problemática a la que se enfrenta el presente trabajo, se abordará todo lo referente a los distintos tipos de señalizaciones de tránsito que son necesarias introducir en el simulador y por consiguiente de las distintas infracciones que pueden surgir, además se hará un análisis de los métodos que se utilizarán para la detección de las infracciones cometidas por el conductor en el simulador.

2.2 Infracciones de conducción.

Es verdaderamente difícil encontrar bibliografía sobre la construcción de simuladores, ya que entre otras cosas estos han estado muy vinculados a los militares en los países que se han atrevido a trabajar en este tema. Sin embargo la gran similitud con los juegos, de los que sí existe bastante bibliografía en el mundo, ha permitido avanzar en la construcción del simulador y por consiguiente ha permitido que se tenga un previo conocimiento de cómo proceder ante una problemática como la que se tiene que resolver en este trabajo. Han servido ideas como la de juegos de autos o incluso de combate o de estrategia que se pueden usar y adaptarlas al Simulador de auto SIMPRO, lo que ha permitido que se acerque cada vez más a un simulador que cumpla con las exigencias del mercado nacional e internacional. En el caso de la Inteligencia Artificial de los juegos lo más que hacen es una simulación de Inteligencia Artificial sin llegar a análisis verdaderamente profundos, en el simulador de auto SIMPRO se necesita llegar un poco más lejos en la inteligencia del sistema sin afectar significativamente la eficiencia del mismo, ya que gráficamente también tiene que ir más lejos que los videojuegos y ya eso implica un mayor consumo de recursos. La primera etapa de búsqueda o de aproximación a la Inteligencia Artificial en el simulador es la detección de infracciones del conductor, la cual podría ser punto de partida para la definición de reglas para los diferentes agentes que intervienen en el simulador. Es decir, mostrarle al conductor cuando comete alguna infracción de tránsito, pero sin llegar a la necesidad de tener que utilizar algoritmos de inteligencia artificial, para lograr este objetivo se utilizaron algunas ideas que fueron extraídas del estudio de algunos juegos virtuales, como puede ser la división del entorno en áreas [Steve, 2002], para una eficiente manera de chequear el ambiente, en algunos casos se realizará una aproximación a la teoría de grafos pero sin profundizar en este tema, tema en el cual se profundizará cuando sí se construya un módulo para prevenir las infracciones de los agentes inteligentes que intervienen en el simulador.

La Herramienta del Simulador de auto SIMPRO se encarga del manejo de las estructuras visuales del simulador, el módulo de detección de infracciones del conductor se adiciona a esta herramienta para así cumplir con uno de los objetivos propuestos.

La Herramienta del Simulador de auto SIMPRO se encarga de tener el estado del entorno. Además lleva toda la información del simulador, los valores que brindan los sensores, como información de cada pedal, de las luces, incluidos los intermitentes, también el ángulo de giro del timón, la posición actual, entre otras variables importantes de las cuales se conocerá en siguientes epígrafes. Esta herramienta se encarga además del control de las colisiones, por lo que solo se necesita conocer en cada instante de tiempo si ocurrió colisión, si este es el caso se incluiría como una infracción, lo cual permite no tener que desarrollar un método para la detección de ese tipo de infracción sino hacer uso de esa información que proviene de la Herramienta del simulador.

Las infracciones en el simulador se pueden clasificar en dos tipos, las infracciones de tránsito y las infracciones mecánicas, a continuación se exponen los aspectos más significativos de cada una de estas para su mejor comprensión.

2.2.1Infracciones de tránsito.

En cada país hay señalizaciones diferentes de tránsito y determinadas señales y leyes particulares. Por esta razón se tomó como base algunas posibles infracciones universales de tránsito y que son necesarias incluir en el entorno virtual del simulador, se han aglutinado un grupo de infracciones que son las que siempre se pueden cometer, pero es uno de los objetivos propuestos la realización de un módulo genérico con posibilidades de incorporar nuevas infracciones sin mucho esfuerzo, ni teniendo que realizar grandes modificaciones en lo ya implementado.

Algunas de estas infracciones son de las más comunes para el conocimiento del conductor, pero no se descuidan otras que en alguna medida influyen en los accidentes de tránsito en Cuba y en el resto del mundo, estas nuevas infracciones formarían parte de la ampliación de este trabajo o quizás de nuevos trabajos que tomen como base todo lo desarrollado hasta el momento en el simulador.

Las infracciones fundamentales y necesarias en el simulador son las siguientes:

- **Pare:** No detenerse ante la señal del pare.
- **Semáforos:** No detenerse cuando está encendida la luz roja del semáforo.
- **Velocidad máxima:** Exceder una velocidad máxima permitida.
- **No parqueo:** Detenerse en un área donde el auto no puede estacionarse.
- **Giro incorrecto:** Girar en una dirección no permitida.
- **Intermitentes:** No encender los intermitentes o hacerlo de manera incorrecta.
- **Peatonal:** No detenerse ante un área de paso peatonal.

Las infracciones antes mencionadas son las más generales y necesarias para comenzar a incorporar la detección de infracciones en el simulador. Existen otras infracciones, las cuales se implementarán a medida que se desarrolle el módulo. Estas restantes infracciones tienen un comportamiento similar a las infracciones antes mencionadas, por lo que con ello se creará un módulo extensible para cualquier tipo de infracción.

2.2.1.1 Los Parámetros.

Para lograr la detección de infracciones de tránsito se necesitan varias informaciones provenientes del entorno virtual, de carácter general informaciones que tienen que ver con el auto y en ocasiones otras informaciones que necesariamente no están relacionadas con él, como puede ser la luz en cada momento que tienen los semáforos, etc. En cada instante de tiempo es necesario conocer toda la información proveniente del auto, tales como, la posición del auto, la velocidad, si tiene encendido algún intermitente, el ángulo de giro del timón, etc. Por ejemplo, para saber si el conductor excedió un límite de velocidad permitida solo se necesita la velocidad del auto, pero para saber si no encendió el intermitente además de la información de los intermitentes se necesita el ángulo de giro o quizás otra información que sería de utilidad para este tipo de infracción. Por tanto es muy variable la cantidad de información necesaria en cada momento.

Para dar solución a lo antes expuesto se creará una clase “SP_CTCar”, la cual tendrá como atributos todas las variables necesarias y que tienen que ver precisamente con las informaciones provenientes del auto. Desde la Herramienta del Simulador de auto SIMPRO se recibirán los parámetros necesarios para crear un objeto de dicha clase, dicho objeto será utilizado por el método responsable de la detección de infracciones. Más adelante se explicará detalladamente como se analizarán las infracciones. Esta clase “SP_CTCar’” incluso pudiera tener algún método que realice algún tipo de cálculo necesario y que facilite el trabajo directamente en el resto de las clases del módulo. Además para el desarrollo del trabajo se necesitará una retroalimentación del entorno, informaciones como la posición de cada señal de tránsito, etc., en el próximo epígrafe se conocerá como se accederá a esta información.

2.2.1.2 Las áreas.

En disímiles artículos referentes a juegos se hace referencia a la subdivisión del entorno virtual en áreas para facilitar la organización del trabajo [Steve, 2002]. En la mayoría de los casos para juegos se divide toda la pista de carrera en áreas, en el caso del entorno virtual del simulador sería la calle, pero no se tendría necesidad de dividir toda una calle para el chequeo de las infracciones sino solo una parte de estas.

Tomando estas ideas se podría dividir todo el entorno virtual en áreas para determinar dentro de cada una de las mismas, qué infracción analizar. Las áreas pudieran estar enmarcadas en una porción de una cuadra, o en algunos casos en intercepciones de ellas. En cada área antes mencionada se chequearía la posible infracción que el conductor podría cometer. Sin embargo esto llevaría a que quizás se tuvieran áreas donde no se analiza ninguna señal de tránsito. Suponga una recta grande donde no se analizaría ninguna de las infracciones, esto sin lugar a dudas puede mejorarse para hacer un chequeo más eficiente de todo el entorno, se podría mejorar de disímiles maneras, se tiene previsto desarrollar mediante una colección donde se almacenen todas las áreas en las cuales se pueda cometer infracción y constantemente preguntar si el auto se encuentra dentro de alguna de estas áreas, en caso de que no se encuentre no se procede a analizar la infracción, en caso contrario se accedería a verificar la

posible infracción a cometer, se profundizará más en este aspecto cuando se analicen las infracciones.

Por otra parte se necesita conocer la posición de cada señal en el entorno, quizás no la posición exacta pero si aproximada, pero con solo la posición de la señal no siempre se resuelve este problema. Por ejemplo, en el caso de una señal de pare el conductor puede detenerse hasta dos o tres metros antes de la señal o simplemente parar casi al lado de la misma, por tanto hay que analizar si se detuvo en toda un área que se tendrá que definir, o en el caso de no exceder una máxima velocidad se analizaría en toda un área donde el conductor no puede sobrepasar dicha velocidad. Por tanto, para saber si en un momento determinado el auto se encuentra cerca de una señal de tránsito se tendrá que definir un área donde se analice si se ha cometido o no la infracción.

En el momento en que se pensó integrar el módulo a la Herramienta del simulador de auto SIMPRO, se estaba desarrollando un entorno sin puentes ni túneles, razón por la cual para facilitar el trabajo se trabajaron estas áreas en dos dimensiones es decir se guardaron 4 puntos que representan las esquinas del área en el plano, cada punto era bidimensional, a medida que se fue avanzando en el desarrollo del entorno virtual se incorporaron algunos puentes y túneles por lo que hubo necesidad de tener que crear estas áreas con la característica de que cada punto fuese tridimensional.

Como aproximación a la teoría de grafos que se mencionó anteriormente, se optó por no utilizar esta estructura de datos, sino simular dicha estructura en caso de que se necesite utilizar, para ello las áreas constarán inicialmente de un identificador el cual se utilizará para diferenciar e identificar las áreas, los tipos de áreas que necesiten tener alguna relación con sus áreas adyacentes constarán de una colección con los identificadores de sus áreas vecinas o adyacentes.

Para poder asociar las áreas con sus respectivas infracciones y con el objetivo de hacer extensible el módulo a otros tipos de infracciones que en el futuro pueden surgir, se concibió que lo idóneo sería definir un conjunto de áreas que fueran del tipo de infracción con que ellas están

asociadas, para que se entienda mejor, podría existir una “SP_CCriticAreaSpeed” la cual sería un área que está asociada al tipo de infracción *control de velocidad máxima* y tendría como atributo la velocidad permitida en esa porción del entorno virtual, así podría existir una “SP_CCriticAreaSemaPhore” asociada a las infracciones de tipo semáforo, y así sucesivamente para cada tipo de infracción. Todos los posibles tipos de áreas asociadas con infracciones que se pudieran definir tienen en común que todas constan de 4 vértices y de un identificador que las hace únicas, para agrupar estos atributos comunes se definió una clase “SP_CCriticArea”.

Los distintos tipos de áreas definidas en el simulador se describen a continuación brevemente:

Tipos de Áreas:

- **Área de tipo Velocidad**: áreas asociadas a infracciones de tipo máxima velocidad permitida, estas áreas tienen como aspecto distintivo un atributo que representa la velocidad máxima, dicho atributo se utilizará para detectar este tipo de infracción, cuando el auto ingrese en un área de este tipo se analizará si cometió la infracción.
- **Área de tipo Semáforo**: áreas asociadas a infracciones de tipo control de semáforo, estas áreas tienen como aspecto distintivo un atributo que representa el semáforo con que ellas están relacionadas, dicho atributo se utilizará para detectar este tipo de infracción. Cuando el auto ingrese en un área de este tipo se analizará si cometió la infracción.
- **Área de tipo Giro**: áreas asociadas a infracciones de tipo giro, además este tipo de áreas es de utilidad en la detección de las infracciones relacionadas con el uso indebido de los intermitentes, estas áreas tienen como aspecto distintivo una colección de enteros donde cada elemento de esta colección es un identificador de una área adyacente a la misma, con este atributo se podrá conocer cuando el auto ingresa en un área que no es una de sus adyacentes o que es lo mismo que realice un giro incorrecto. Cuando el auto ingrese en un área de este tipo se analizará si cometió la infracción.
- **Área de tipo Mixta**: áreas asociadas a infracciones de distintos tipos, este tipo de infracciones no poseen un atributo distintivo necesario para detectar su infracción sino que

con los atributos provenientes del auto se podrían detectar la infracción correspondiente, entre estas áreas se encontrarán las asociadas a infracciones tales como infracciones de tipo pare, de tipo paso peatonal y de tipo no parqueo, estas áreas tienen como aspecto distintivo tres atributos booleanos, los cuales representan la infracción a tratar**,** en dependencia del valor que ellos tengan será la infracción que se analizará, estos atributos podrían ser sustituidos por una colección donde sus atributos fueran de tipo booleano donde cada posición del arreglo significa el identificador de la infracción y si el valor está en verdadero es que se analiza dentro de esa área esa infracción, en fin sea cual sea el método para representar la infracción a tratar, dicho método se utilizará para detectar ese tipo de infracción, cuando el auto ingrese en un área de este tipo se analizará si cometió la infracción del tipo especificada por su atributo.

Todas las áreas que se necesiten para controlar las infracciones en el simulador, serán guardadas en un fichero, desde la Herramienta del Simulador de auto SIMPRO se cargarán las áreas previamente guardadas en el fichero y se almacenarán en una colección, este proceso se realiza cada vez que comienza la simulación virtual.

Para la detección de las infracciones antes mencionadas se concibió el uso de una clase denominada “SP_CTInfraction” la cual tiene como función el control sobre los objetos antes descritos, ya sea las áreas o el auto, además tendrá un gran número de atributos necesarios para poder detectar si el conductor comete cualquier tipo de infracción de tránsito. Los métodos fundamentales para la detección de las infracciones se describen a continuación:

- **LoadFile:** Método necesario para cargar las áreas del fichero, dichas áreas se irán adicionando a una colección de áreas que permitirá en su momento identificar en el área en la cual se encuentra situado el auto y así proceder a detectar la infracción correspondiente a cada tipo de área.
- **InfractionType:** Método necesario para detectar y por consiguiente clasificar las infracciones que pudieran cometerse, este método utilizará un atributo fundamental que

sería el auto, objeto de tipo “SP_CTCar” con el cual se tendrá acceso a los parámetros como la velocidad, ángulo de giro, etc. este método llamará a otros métodos que son los encargados de verificar si ocurrió o no infracción.

Hasta el momento se ha hecho énfasis que las áreas están asociadas a las infracciones que se pueden cometer dentro de ellas. Pero existen determinadas infracciones que pueden cometerse en cualquier parte del entorno virtual o se necesitan analizarse fuera del área enmarcada, por ejemplo:

- **Pare:** Mientras el auto está dentro del área de pare el conductor puede detenerse y no habrá cometido la infracción, razón por la cual es necesario verificar la infracción en el momento que el auto abandone el área de tipo pare y entonces chequear que si nunca se detuvo dentro de ella es que cometió la infracción.
- **Intermitentes incorrectos:** En cualquier posición del entorno virtual el conductor puede proceder a encender los intermitentes sin razón alguna. En este caso incurrirá en una infracción.
- **Semáforo:** Mientras el auto está dentro del área de tipo semáforo el conductor pudo detenerse o no mientras estuvo encendida la luz roja, si se detuvo no habrá cometido la infracción, razón por la cual es necesario verificar la infracción en el momento que abandone el área y entonces chequear que si nunca se detuvo dentro del área es que cometió la infracción.

2.2.2Infracciones mecánicas.

El Modelo matemático utilizado en el Simulador SIMPRO está siendo sometido a continuos cambios, razón por la cual en el presente trabajo sólo se exponen ideas que podrían ser utilizadas en la continuación de este u otros trabajos, los cuales podrían analizar y continuar la propuesta que aquí se presenta para la detección de las infracciones mecánicas.

Las Infracciones mecánicas son aquellas que están relacionadas con las ejecuciones mecánicas que haga el conductor incorrectamente como pueden ser “Cambio de velocidad incorrecto” o “No arrancar en la primera velocidad” o quizás “No tener el pedal del embrague presionado al hacer el cambio de velocidad”, etc. Este tipo de infracciones son tratadas de una manera diferente a las infracciones de conducción, ya que no se necesita del chequeo del entorno para detectarlas, sino que hay que introducirse en el Modelo matemático y chequear varias variables que son necesarias para este fin, estas variables se conocerá en el próximo epígrafe.

Algunas de las infracciones mecánicas que se pueden detectar son las siguientes:

- **Primer cambio incorrecto**: Si el auto para comenzar a desplazarse lo hace con una velocidad que no sea *la primera* el carro se apaga.
- **Arranque con velocidad**: Si se enciende el motor del auto con algún cambio en la caja de velocidad diferente del neutral se produce un movimiento brusco del auto.
- **Salida incorrecta**: Si cuando el auto comienza a avanzar con la primera velocidad puesta y el conductor saca el pie del embrague de forma muy rápida el auto puede apagarse, además si el conductor acelera el auto de forma brusca se rastrillan las gomas.
- **Salida con freno de emergencia activado:** Si el auto sale en primera con la emergencia puesta el carro hace un ruido extraño y el conductor no puede poner en marcha el auto.
- **Cambio incorrecto de velocidad**: Cuando el auto va en una velocidad acelerando y el conductor se equivoca y pone un cambio inmediato inferior el auto se frena bruscamente, de lo contrario si la velocidad va disminuyendo y el cambio es superior se incurre en dicha infracción.
- **Cambiar sin embrague:** Aplicar un cambio de la caja de velocidad sin presionar el embrague.
- **Frenar sin embrague:** Para frenar el conductor tiene que apretar el embrague y el freno a la vez, ya que si solo aprieta el embrague el carro realiza un brinco y se apaga.

- **Motor apagado involuntariamente**: Siempre que se cometa un error que tenga como consecuencia que se apague el motor inesperada o involuntariamente se incurre en una infracción.

2.2.2.1 Los Parámetros.

Para lograr la detección de infracciones mecánicas se necesitan varias informaciones provenientes del Modelo matemático, dichas informaciones estarían aglutinadas en una clase denominada “SP_CMCar”, la cual tendrá como atributos todas las variables necesarias y que tienen que ver precisamente con las informaciones como antes se mencionó provenientes del Modelo matemático. Desde la Herramienta del simulador se recibirán los parámetros necesarios para crear un objeto de dicha clase, dicho objeto será utilizado por el método responsable de la detección de infracciones mecánicas. A continuación se exponen los principales parámetros, así como los valores que los mismos pueden tomar:

- **Clutch o embrague:**
  Valor: presionado o no presionado (cambio gradual).
- **Caja de velocidad:**
  Valor: neutral, 1ra, 2da, 3ra, 4ta, 5ta, 6ta (opcional).
- **Emergencia:**
  Valor: puesta o no.
- **Acelerador:**
  Valor: presionado o no presionado (cambio gradual).
- **Motor:**
  Valor: encendido o apagado.
- **Aceleración:**
  Valor: positivo si aumenta la velocidad, negativo si disminuye.

2.3 Conclusiones.

En el transcurso de este capítulo se realizó un análisis profundo de los tipos de infracciones que se tienen en cuenta en el simulador, se acercó al lector al conocimiento básico de las infracciones y de los métodos que se utilizarán para la detección de las mismas. Con respecto a las infracciones mecánicas se planteó una propuesta de cómo pueden ser detectadas este tipo de infracciones y se recomendó un grupo de variables que se necesitan obtener del Modelo matemático para poder detectar este tipo de infracciones.

Capítulo 3 : TENDENCIAS Y TECNOLOGÍAS ACTUALES A DESARROLLAR.

3.1 Introducción.

En el presente capítulo se expone el objeto de estudio que sustenta la presente investigación, se especifican detalladamente las labores que serán objeto de automatización para determinar de forma clara el límite y responsabilidad del sistema a construir así como las Herramientas y metodologías a utilizar. Se realiza una descripción del contexto del negocio expresado en un modelo del dominio, se hace un levantamiento de los requisitos funcionales y no funcionales, los requisitos funcionales se estructuran mediante los Casos de Uso del sistema, de los cuales se ofrece una descripción textual.

3.2 Objeto de estudio.

3.2.1Situación problémica.

En el simulador de auto SIMPRO no se cuenta con un sistema en el cual se detecten las distintas infracciones de tránsito y mecánicas realizadas por el conductor del simulador, esto imposibilita que el sistema cuente con el realismo deseado, pero además necesita que los usuarios cuenten con un medio a través del cual conozcan las infracciones o ilegalidades que cometieron en su paseo virtual, además de los posibles accidentes que de esta forma pueden ser prevenidos antes de que el conductor vaya a la práctica real de conducción.

3.2.2Problema.

Teniendo en cuenta lo antes planteado nuestro principal problema está en ¿Cómo detectar y clasificar las infracciones de tránsito y mecánicas cometidas por el conductor del simulador de auto SIMPRO?

3.3 El proceso Unificado de Desarrollo de Software (RUP) como base en el desarrollo de la solución propuesta.

El Proceso Unificado de Desarrollo de Software es un proceso de desarrollo propuesto por "Rational Software Corporation" resultado del esfuerzo de su experiencia en el desarrollo de software y de sus autores Ivar Jacobson, Grady Booch y James Rumbaugh. (Rational Software Corporation).[Jacobson, 2000]

Jacobson, Booch y Rumbaugh (1999) mencionan que RUP es un proceso de ingeniería de software. El cual proporciona un enfoque disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo de software. De acuerdo con Rational, es una metodología de desarrollo orientada a objetos y habilitada para Web. RUP provee un curso de acción, plantillas y ejemplos para los aspectos y las etapas de desarrollo de software. RUP son herramientas de ingeniería de software que combinan los aspectos de desarrollo tales como

etapas definidas, técnicas y prácticas con otros componentes de desarrollo tales como documentos, modelos, manuales y código dentro de un entorno unificado.

RUP establece cuatro fases de desarrollo, cada una de las cuales está organizada en un cierto número de iteraciones separadas que deben satisfacer criterios definidos antes de que se tome la siguiente fase:

- **Fase de inicio o concepción:** Los desarrolladores definen el enfoque del proyecto y el caso de uso.
- **Fase de elaboración:** Los desarrolladores analizan las necesidades del proyecto a profundidad, definen sus características y establecen la arquitectura.
- **Fase de construcción:** Los desarrolladores crean el diseño de la aplicación y el código fuente.
- **Fase de transición:** Los desarrolladores liberan el sistema para los usuarios. RUP proporciona un prototipo cada vez que termina cada iteración.

Características principales de RUP.

RUP posee disímiles características, pero los verdaderos aspectos definitorios del Proceso Unificado se resumen en tres frases claves- dirigido por casos de uso, centrado en la arquitectura, e iterativo e incremental. Esto es lo que hace único el Proceso Unificado.

Aspectos definitorios:

- **Guiado por casos de uso:** La razón de ser de un sistema de software es servir a los usuarios ya sean humanos u otros sistemas; un caso de uso es una facilidad que el software debe proveer a sus usuarios. Los casos de uso reemplazan la antigua especificación funcional tradicional y constituyen la guía fundamental establecida para las actividades a realizar durante todo el proceso de desarrollo incluyendo el diseño, la implementación y las pruebas del sistema.

- **Centrado en arquitectura:** La arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros por plataformas de software, sistemas operativos, manejadores de bases de datos, protocolos, consideraciones de desarrollo como sistemas heredados y requerimientos no funcionales. Los casos de uso guían el desarrollo de la arquitectura y la arquitectura se retroalimenta en los casos de uso, los dos juntos permiten conceptualizar, gestionar y desarrollar adecuadamente el software.
- **Iterativo e Incremental:** Para hacer más manejable un proyecto se recomienda dividirlo en ciclos. Para cada ciclo se establecen fases de referencia, cada una de las cuales debe ser considerada como un mini proyecto cuyo núcleo fundamental está constituido por una o más iteraciones de las actividades principales básicas de cualquier proceso de desarrollo.

En conclusión, debido a que cada uno de los modelos del ciclo de vida para el desarrollo de software cuenta con características propias, esto hace que dependa de las necesidades del cliente utilizar el modelo adecuado. Sin embargo, el aplicar un modelo iterativo presenta mayores ventajas a los modelos de cascada, y también es aplicable en la mayoría de los proyectos cuya finalidad es la creación de un producto de software.

Por lo anterior, se puede afirmar que la selección de un modelo iterativo de desarrollo de software puede ser un factor determinante en el éxito del proyecto.

3.4 El Lenguaje Unificado de Modelado (UML) como soporte de la modelación de la solución propuesta.

RUP utiliza el lenguaje Unificado de Modelado (UML), UML se ha convertido en un estándar de facto que tiene entre algunas de sus características las siguientes:

- Permite modelar sistemas valiéndose de técnicas orientadas a objetos.
- Permite especificar todas las decisiones de análisis, diseño e implementación, construyéndose así modelos precisos, no ambiguos y completos.

- Puede conectarse con lenguajes de programación (Ingeniería directa e inversa).
- Permite documentar todos los artefactos de un proceso de desarrollo.
- Cubre las cuestiones relacionadas con el tamaño propio de los sistemas complejos y críticos.
- Es un lenguaje muy expresivo que cubre todas las vistas necesarias para desarrollar y luego desplegar los sistemas.
- Existe un equilibrio entre expresividad y simplicidad, pues no es difícil de aprender ni de utilizar.

3.5 Herramientas de desarrollo

Las herramientas de desarrollo que se utilizarán en todas las fases de construcción, son las que se describen a continuación.

3.5.1Visual Studio 2003.

Microsoft Visual Studio 2003 es un software desarrollado por la compañía Microsoft, entre las características fundamentales que presenta se encuentran algunas como, su capacidad de crear software profesional con rapidez, reducir los costos de funcionamiento de tecnologías de la información y además se integra con una amplia gama de aplicaciones, sistemas y dispositivos, la plataforma sobre la que se ha desarrollará este trabajo es el entorno de programación Visual C++, debido a que la programación correspondiente a la parte gráfica y el Modelo matemático se realizan en C++, además de ser un lenguaje con posibilidad de desarrollo en plataforma libre.

3.5.2Visual SourceSafe 2005.

Microsoft Visual SourceSafe es un sistema de control de versiones en el nivel de archivos, desarrollado por la compañía Microsoft.

VSS permite a muchos tipos de organizaciones trabajar en distintas versiones de un proyecto al mismo tiempo. Esta funcionalidad es especialmente ventajosa en un entorno de desarrollo de software, donde se usa para mantener versiones de código paralelas.

VSS admite el desarrollo multiplataforma al permitir la edición y el uso compartido de los datos. Se ha diseñado para controlar los problemas de seguimiento y portabilidad que implica mantener una base de control de código fuente, como una base de código de software, en varios sistemas operativos. Para los desarrolladores, VSS aloja código reutilizable u orientado a objetos. Asimismo, facilita el seguimiento de las aplicaciones que utilizan módulos de código concretos.

VSS incluye, como mínimo, las siguientes funciones:

- Ayuda al equipo a evitar la pérdida accidental de archivos.
- Permite realizar un seguimiento de las versiones anteriores de un archivo.
- Admite la bifurcación, el uso compartido, la combinación y la administración de versiones de archivos.
- Realiza el seguimiento de las versiones de proyectos completos.
- Realiza el seguimiento del código modular (un archivo que se reutiliza, o se comparte, en varios proyectos).

3.5.3 Por qué deberíamos usar herramientas CASE de modelado con UML.

A medida que los sistemas que hoy se construyen se tornan más y más complejos, las herramientas de modelado con UML ofrecen muchos beneficios para todos los involucrados en un proyecto, por ejemplo, administrador del proyecto, analistas, arquitectos, desarrolladores y otros. Las herramientas CASE de modelado con UML nos permiten aplicar la metodología de análisis y diseño orientados a objetos y abstraernos del código fuente, en un nivel donde la arquitectura y el diseño se tornan más obvios y más fáciles de entender y modificar. Cuanto más grande es un proyecto, es más importante utilizar una herramienta CASE. Al usar las herramientas CASE:

- Los Analistas de Negocio/ Sistemas pueden capturar los requisitos del negocio/sistema con un modelo de casos de uso
- Los Diseñadores/Arquitectos pueden producir el modelo de diseño para articular la interacción entre los objetos o los subsistemas de la misma o de diferentes capas (los diagramas UML típicos que se crean son los de clases y los de interacción)
- Los Desarrolladores pueden transformar rápidamente los modelos en una aplicación funcionando, y buscar un subconjunto de clases y métodos y asimilar el entendimiento de cómo lograr interfaces con ellos.

3.5.3.1 Herramienta CASE Enterprise Architect(EA)

Enterprise Architect es una herramienta comprensible de diseño y análisis UML, cubre el desarrollo de software desde el paso de los requerimientos a través de las etapas del análisis, modelos de diseño, pruebas y mantenimiento. EA es una herramienta multi-usuario, basada en Windows, diseñada para ayudar a construir software robusto y fácil de mantener. Ofrece salida de documentación flexible y de alta calidad.

Las bases de Enterprise Architect están construidas sobre la especificación de UML 2.0, además usa Perfiles UML para extender el dominio de modelado, mientras que la Validación del Modelo asegura integridad. Combina Procesos de Negocio, Información y Flujos de trabajo en un modelo.

Características de Enterprise Architect:

- Soporta los 13 diagramas de UML 2.1.
- Soporte para perfil de estilo UML 2.0.
- Interfaz de usuario intuitiva.
- Documentación flexible y comprensible.
- Ingeniería de Código Directa e Inversa.

- Soporte para 'tecnologías agregadas' usando Tecnologías MDG (Generación Dirigida por Modelos).
- Modelado de Base de Datos.
- Soporta Control de Versiones.
- Capacidad para compartir modelos de diversas maneras.
- Soporte de esquema XML.
- Compare la Utilidad.
- Soporta Línea Base.
- Archivos Binarios de Ingeniería Inversa para Java y .NET.
- Soporte en Administración de Requisitos.
- Importación / Exportación de modelos en formato XML.
- Soporta Seguridad de Usuario.
- Soporte para Prueba.
- Soporte para el Mantenimiento.
- Soporte para Administración de Proyecto.
- Soporte para información de estado del sistema.

3.6 Modelo del dominio.

En dependencia de la situación o escenario que se presente, hay varias alternativas para expresar el contexto del sistema en una forma utilizable para los desarrolladores de software, las principales alternativas están en realizar un modelado del dominio o un modelado del negocio. Un modelo del dominio captura los tipos más importantes de objetos que existen o los eventos que suceden en el entorno en el que trabaja el sistema y enlaza estos objetos unos con otros. Con el

desarrollo de un glosario de términos donde se identifiquen y se asignen nombres a los objetos del dominio se logrará un mejor entendimiento entre los clientes y desarrolladores que trabajarán en el sistema.[Larman, 1999]

Se realizará un modelo del dominio debido a las características del sistema a construir, este modelo se integrará al modelo del dominio realizado en la Herramienta del Simulador SIMPRO.

3.6.1Glosario de Términos del dominio.

Área: define un área en la que se analiza algún tipo de infracción de tránsito.

AreaMixed: área en la que se analiza la infracción de tránsito de tipo pare, paso peatonal y no parqueo.

AreaTurn: área en la que se analiza la infracción de tránsito de giro y de tipo intermitente.

AreaSemaphore: área en la que se analiza la infracción de tránsito de tipo semáforo

AreaSpeed: área en la que se analiza la infracción de tránsito de tipo velocidad.

Infracciones de tránsito: infracciones que comete el conductor por su acción personal al conducir, donde el conductor no respeta señalizaciones de tránsito, ejemplo: No detenerse ante una señalización de pare, etc.

Infracciones mecánicas: infracciones relacionadas con las ejecuciones mecánicas que haga el conductor incorrectamente, que a largo plazo afectan el funcionamiento mecánico del auto, estas infracciones son detectadas por variables obtenidas del Modelo matemático, ejemplo: Cambio de velocidad incorrecto, etc.

MCar: agrupa los atributos principales del simulador virtual, necesarios para detectar las infracciones mecánicas y que en su mayoría son extraídos del Modelo matemático, ejemplo: acelerador, caja de velocidad, emergencia, etc.

TCar: agrupa los atributos principales del simulador virtual, necesarios para detectar las infracciones de tránsito, ejemplo: velocidad del auto, ángulo de giro, intermitentes, etc.

Vector: vector tridimensional para la definición de coordenadas.

3.6.2Diagrama de clases del dominio.

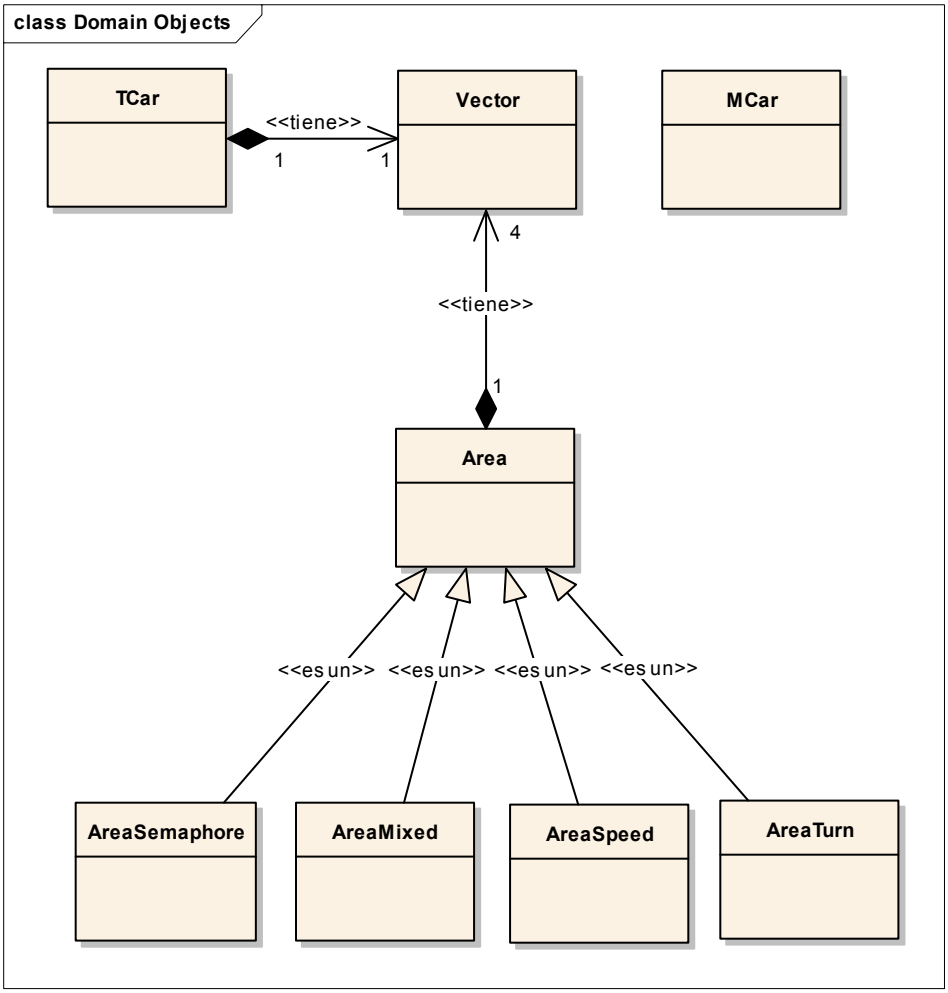

Figura 1: Diagrama del modelo del dominio.

3.7 Reglas del negocio.

Las áreas correspondientes a las señalizaciones de tránsito incluidas en el Entorno Virtual de la Herramienta del simulador deben ser definidas por un diseñador.

Las áreas definidas por el diseñador deben ser guardadas en un fichero.

La carga del fichero de las áreas debe realizarse al iniciar la Herramienta del simulador.

3.8 Captura de Requisitos.

A continuación se expondrán los requisitos funcionales y no funcionales del sistema.

3.8.1 Requisitos no funcionales.

1. **Soporte:** En una versión inicial deberá ser compatible con la plataforma Windows, pero debe estar preparado para que con pequeñas modificaciones pueda migrar al Sistema Operativo Linux.
2. **Legales:** Se regirá por las normas ISO 9000.
3. **Software:** Sistema operativo Windows.
4. **Hardware:** Compatibilidad con tarjetas gráficas de la familia NVIDIA (Geforce 3, Geforce 4, FX 5200).
5. **Diseño e implementación:** Se regirá por la filosofía de Programación Orientada a Objetos y el lenguaje de programación a utilizar tiene que ser el C++.

3.8.2 Requisitos funcionales.

1. **Cargar fichero.**

   1.1 Abrir fichero.

   1.2 Crear Buffer.

1.3 Cargar Datos al Buffer.

1.4 Adicionar Buffer en una colección.

2. Clasificar infracciones de tránsito.

2.1 Crear vector.

2.2 Actualizar auto de tránsito.

2.3 Buscar las áreas en las que se encuentra el auto.

2.3.1 Determinar si el auto se encuentra dentro de un área.

2.3.2 Determinar tipo de área en la que se encuentre el auto.

2.3.3 Insertar áreas donde se encuentra el auto en una colección de áreas.

2.4 Detectar infracciones que se cometen fuera de las áreas.

2.5 Detectar infracciones que se cometen dentro de las áreas.

2.6 Actualizar colección de contadores de infracciones de tránsito.

2.7 Destruir vector.

3. Clasificar infracciones mecánicas.

3.1 Actualizar auto mecánico.

3.2 Detectar infracciones mecánicas.

3.3 Actualizar colección de contadores de infracciones mecánicas.

4. Restablecer infracciones de tránsito.

5. Restablecer infracciones mecánicas.

3.9 Modelos de Casos de Uso del sistema.

El modelo de casos de uso permite a los desarrolladores de software y los clientes lleguen a un acuerdo sobre los requisitos, es decir sobre las condiciones y posibilidades que debe cumplir el sistema. El modelo de casos de uso proporciona la entrada fundamental para el análisis, el diseño y las pruebas. [Jacobson, 2000]

El modelo de casos de uso es un modelo del sistema que contiene actores, casos de uso y sus relaciones.

3.9.1Identificación de los actores.

Los actores se definen como los roles que pueden desempeñar terceros fuera del sistema, que colaboran con el sistema, pueden ser personas, sistemas externos, máquinas, etc.

Tabla 1: Definición de actores.

Actores	Justificación
Programador	Es el que se beneficiará con las funcionalidades que brinda el Módulo para la detección de infracciones de tránsito del conductor en el simulador de auto, de modo general le permitirá conocer el tipo y cantidad de infracciones que cometa el conductor en el simulador.

3.9.2Listado de Casos de Uso.

Tabla 2: Descripción de los casos de uso del sistema.

CU-1	Cargar fichero
Actor	Programador
Descripción	Este caso de uso le brinda la posibilidad al programador de cargar las áreas de un fichero, el programador invoca la carga del fichero, el sistema procede a verifica la existencia del fichero, se abre el fichero, se leen los datos y se almacena la información en un buffer mientras no se llegue al fin del fichero, este información a medida que es extraída se inserta en una colección. De ocurrir algún error en el proceso se le informa al programador.
Referencia	R 1, R 1.1, R 1.2, R 1.3, R1.4
CU-2	Clasificar infracciones de tránsito.
Actor	Programador
Descripción	Este caso de uso es el que brinda la posibilidad al programador de clasificar las infracciones de tránsito, el programador invoca la clasificación de infracciones, para ello se crea un vector con sus respectivos parámetros, luego se actualiza el auto con los parámetros recibidos, entre ellos el vector anteriormente creado, por último, el sistema procede a busca las áreas en las que se encuentra el auto, a continuación se procede a detectar las infracciones que se cometen cuando el auto no se encuentra dentro de alguna área, para finalizar, si el auto está dentro de algún área, el sistema procede a detectar si

	ocurre infracción dentro de las áreas que se encuentra, en los casos en que el sistema procedió a analizar las infracciones en caso de que ocurra la infracción se actualiza el contador de infracción en la posición que corresponde a dicha infracción en la colección de infracciones cometidas.
Referencia	R 2, R 2.1, R 2.1.1, R 2.2, R 2.3, R 2.3.1, R 2.3.2, R 2.3.3, R 2.4, R 2.5, R 2.6, 2.7
CU-3	Clasificar infracciones mecánicas.
Actor	Programador
Descripción	Este caso de uso es invocado por el programador, el caso de uso le brinda la posibilidad al programador de clasificar las infracciones mecánicas, el programador invoca la clasificación de infracciones, para ello se actualiza el auto con sus respectivos parámetros, se finaliza cuando el programador invoca la detección de infracciones mecánicas y el sistema procede a detectar dichas infracciones.
Referencia	R 3, R 3.1, R 3.2, R 3.3
CU-4	Restablecer infracciones de tránsito
Actor	Programador
Descripción	Este caso de uso es invocado por el programador, el caso de uso le brinda la posibilidad al programador de restablecer las cantidades de infracciones de tránsito cometidas por el conductor a 0, el programador invoca al sistema solicitando que establezca el valor de cada elemento

	de la colección con las cantidades de infracciones a 0, el sistema procede a recorrer cada elemento de la colección y establecer su valor a 0.
Referencia	R 4
CU-5	Restablecer infracciones mecánicas
Actor	Programador
Descripción	El caso de uso es invocado por el programador, el caso de uso le brinda la posibilidad al programador de restablecer las cantidades de infracciones mecánicas cometidas por el conductor a 0, el programador invoca al sistema solicitando que establezca el valor de cada elemento de la colección con las cantidades de infracciones a 0, el sistema procede a recorrer cada elemento de la colección y establecer su valor a 0.
Referencia	R 5

3.9.3Diagrama de Casos de usos del sistema

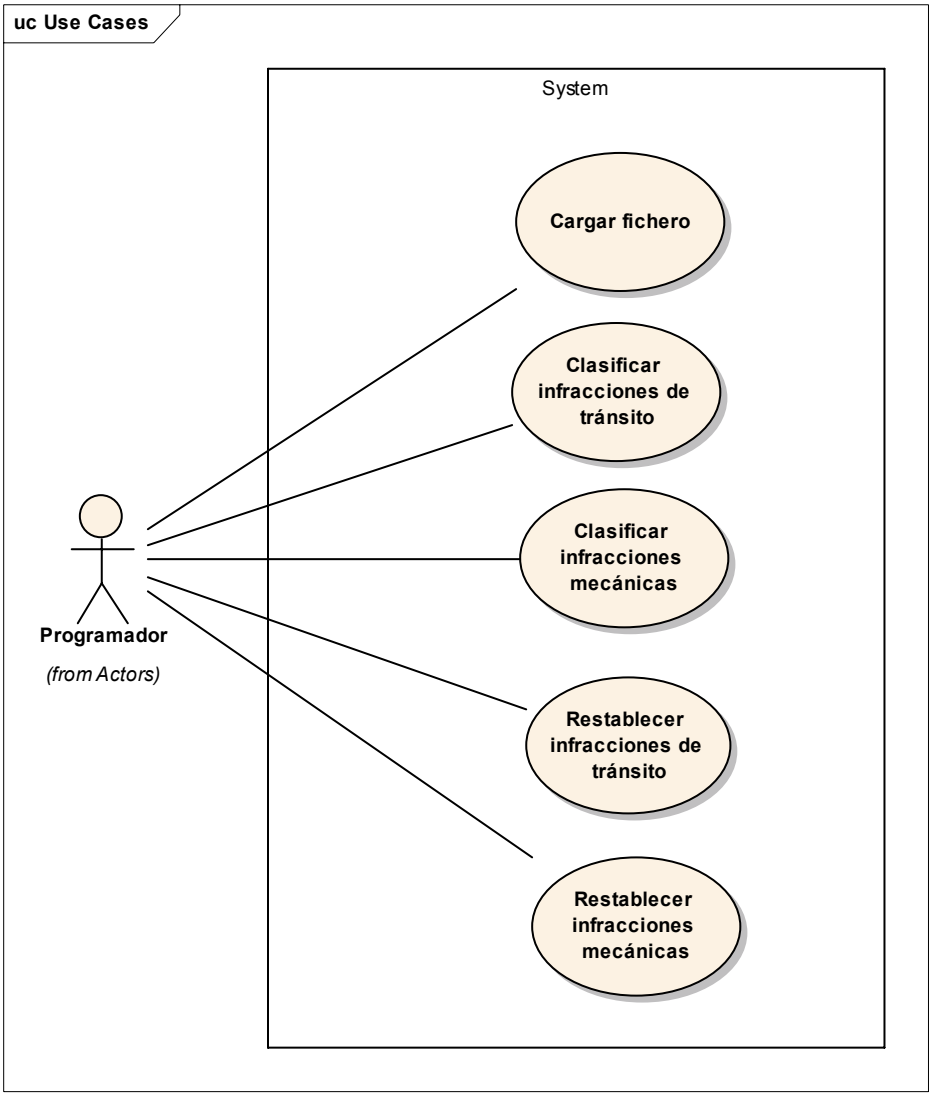

Figura 2: Diagrama de CUS.

3.9.4Casos de Uso expandidos.

La expansión de los casos de uso se realiza con el objetivo de comprender la funcionalidad de cada caso de uso, ya que no basta con una representación gráfica mediante el Diagrama de casos de uso.

Tabla 3: CU del sistema “Cargar fichero”.

Nombre del caso de uso	Cargar fichero
Actores	Programador
Propósito	Permite cargar datos del fichero y almacenarlos en una colección.
Resumen: Este caso de uso es el que brinda la posibilidad al programador de cargar las áreas de un fichero, el programador invoca la carga del fichero, el sistema procede a verifica la existencia del fichero, se abre el fichero, se leen los datos y se almacena la información en un buffer mientras no se llegue al fin del fichero, este información a medida que es extraída se inserta en una colección. De ocurrir algún error en el proceso se le informa al programador.	
Referencias	R1, R 1.1, R 1.2, R 1.3, R1. 4
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1- El programador invoca la carga del fichero.	
	2- El sistema procede a abrir el archivo.
	3- El sistema procede a la lectura de los datos y los almacena en el buffer indicado.

	4- El sistema procede a insertar el buffer con los datos extraído del fichero en la colección.
	5- Se devuelve la validez de las operaciones.
Flujo alternativo	
	2.1- Si surge un error al abrir el archivo ir al paso 5.
	3.1- Si surge un error al leer los datos del archivo ir al paso 5.
Precondiciones	El diseñador tuvo que haber definido y almacenado las áreas en el fichero.
Poscondiciones	Quedan almacenadas las áreas extraídas del fichero en una colección.

Tabla 4: CU del sistema “Clasifica infracciones de tránsito”.

Nombre del caso de uso	Clasificar infracciones de tránsito
Actores	Programador
Propósito	Permite determinar la cantidad y el tipo de infracciones de tránsito cometidas por el conductor.

Resumen: Este caso de uso es el que brinda la posibilidad al programador de clasificar las infracciones de tránsito, el programador invoca la clasificación de infracciones, para ello se crea un vector con sus respectivos parámetros, luego se actualiza el auto con los parámetros recibidos, entre ellos el vector anteriormente creado, por último, el sistema procede a buscar las áreas en las que se encuentra el auto, a continuación se procede a detectar las infracciones que se cometen cuando el auto no se encuentra dentro de alguna área, para finalizar, si el auto está dentro de algún área, el sistema procede a detectar si ocurre infracción dentro de las áreas que se encuentra, en los casos en que el sistema procedió a analizar las infracciones en caso de que ocurra la infracción se actualiza el contador de infracción en la posición que corresponde a dicha infracción en la colección de infracciones cometidas.	
Referencias	R 2, R 2.1, R 2.1.1, R 2.2, R 2.3, R 2.3.1, R 2.3.2, R 2.3.3, R 2.4, R 2.5, R 2.6, 2.7
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
Sección: Crear Vector.	
1- Se solicita la creación de un vector pasándole sus tres parámetros.	
	2- Se crea un vector.
Sección: Actualizar auto de tránsito.	

1- Se solicita la actualización del auto pasando los parámetros, entre ellos el vector anteriormente creado.	
	2- Se actualiza el auto.
Sección: Detectar infracciones de tránsito.	
1-El programador invoca la clasificación de infracciones tránsito.	
	2- El sistema procede a busca las áreas en las que se encuentra el auto.
	3- El sistema procede a detectar las infracciones que se cometen cuando el auto no se encuentra dentro de algún área.
	4- Si el auto está dentro de algún área, el sistema procede a detectar si ocurre infracción dentro de las áreas que se encuentre el auto.
Flujo alternativo	
	3.1- Si ocurre la infracción se actualiza el contador de infracción en la posición que

	corresponde a dicha infracción en la colección de infracciones cometidas.
	4.1- Si ocurre infracción y es del tipo de las que no se necesita información cuando el auto abandona el área, se actualiza el contador de infracción en la posición que corresponde a dicha infracción en la colección de infracciones cometidas.
Precondiciones	----------
Poscondiciones	Se obtienen las cantidades de infracciones de tránsito cometidas clasificadas por tipos.

Tabla 5: CU del sistema “Clasifica infracciones mecánicas”.

Nombre del caso de uso	Clasifica infracciones mecánicas
Actores	Programador
Propósito	Permite conocer la cantidad y el tipo de infracciones mecánicas cometidas.
Resumen: Este caso de uso es invocado por el programador, el caso de uso le brinda la posibilidad al programador de clasificar las infracciones mecánicas, el programador invoca la clasificación de infracciones, para ello se actualiza el auto con sus respectivos parámetros, se finaliza cuando el programador invoca	

la detección de infracciones mecánicas y el sistema procede a detectar dichas infracciones.	
Referencias	R3, R 3.1, R 3.2, R 3.3
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
Sección: Actualizar auto mecánico.	
1- Se solicita la actualización del auto pasando los parámetros necesarios.	
	2-El sistema procede a la actualización del auto.
Sección: Detectar infracciones mecánicas.	
1-El programador solicita la detección de infracciones mecánicas.	
	2-El sistema procede a detectar las infracciones mecánicas.
Flujo alternativo	
	2.1- Si ocurre la infracción se actualiza el contador de infracción en la posición que corresponde a dicha infracción en la

	colección de infracciones cometidas.
Precondiciones	----------
Poscondiciones	Se obtienen las cantidades de infracciones mecánicas cometidas clasificadas por tipos.

Tabla 6: CU del sistema “Restablecer infracciones de tránsito”.

Nombre del caso de uso	Restablecer infracciones de tránsito
Actores	Programador
Propósito	Permite restablecer las cantidades de infracciones de tránsito cometidas en 0.
Resumen: Este caso de uso es invocado por el programador, el caso de uso le brinda la posibilidad al programador de restablecer las cantidades de infracciones de tránsito cometidas a 0, el programador invoca al sistema solicitando que establezca el valor de cada elemento de la colección que contiene las cantidades de infracciones a 0, el sistema procede a recorrer cada elemento de la colección y establecer su valor a 0.	
Referencias	R4
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1-El programador invoca al sistema	

solicitando que establezca el valor de cada elemento de la colección con la cantidad de infracciones de tránsito cometidas en 0.	
	2-El sistema procede a recorrer cada elemento de la colección y establece su valor en 0.
Precondiciones	----------
Poscondiciones	Las cantidades de infracciones de tránsito cometidas quedan reiniciadas.

Tabla 7: CU del sistema “Restablecer infracciones mecánicas”.

Nombre del caso de uso	Restablecer infracciones mecánicas
Actores	Programador
Propósito	Permite restablecer las cantidades de infracciones mecánicas cometidas en 0.
Resumen: Este caso de uso es invocado por el programador, el caso de uso le brinda la posibilidad al programador de restablecer las cantidades de infracciones mecánicas cometidas a 0, el programador invoca al sistema solicitando que establezca el valor de cada elemento de la colección que contiene las cantidades de infracciones a 0, el sistema procede a recorrer cada elemento de la colección y establecer su valor a 0.	

Referencias	R5	
Curso normal de los eventos:		
Acción del actor		**Respuesta del sistema**
1-El programador invoca al sistema solicitando que establezca el valor de cada elemento de la colección con la cantidad de infracciones mecánicas cometidas en 0.		
		2-El sistema procede a recorrer cada elemento de la colección y establece su valor en 0.
Precondiciones	----------	
Poscondiciones	Las cantidades de infracciones mecánicas cometidas quedan reiniciadas.	

3.10 Conclusiones.

En este capítulo se comenzó a desarrollar la propuesta de solución, para ello se trató la situación problémica y el problema que da origen al desarrollo de la solución. Se definieron los requisitos que debe cumplir el sistema en su totalidad, así como los actores del sistema, donde se incluyó una amplia descripción de cada una de sus funcionalidades. Como resultado de todo lo antes expuesto se está en condiciones de ingresar en la construcción del sistema.

Capítulo 4 : Construcción de la solución propuesta.

4.1 Introducción.

En el presente capítulo se plantea el diseño del sistema, se muestra los principales artefactos UML obtenidos en los flujos de trabajo de diseño e implementación. Se definen los diagramas de interacción, diagrama de clases de diseño, diagrama de componentes y de despliegue.

La nomenclatura utilizada en los diagramas de clases, se puede consultar en el epígrafe “Estándares de codificación”, y finalmente, se realiza el modelo de despliegue y el modelo de implementación para una mejor descripción de la solución.

4.2 Diagramas de clases de diseño

El Diagrama de Clases es el diagrama principal para el análisis y diseño. Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia. La definición de clase incluye definiciones para atributos y operaciones. El modelo de casos de uso aporta información para establecer las clases, objetos, atributos y operaciones. [Jacobson, 2000]

A continuación se ilustran los diagramas de clases correspondientes a los casos de uso Clasificar infracciones tránsito y Clasificar infracciones mecánicas, los restantes casos de uso, hacen uso de estos dos diagramas de clases del diseño.

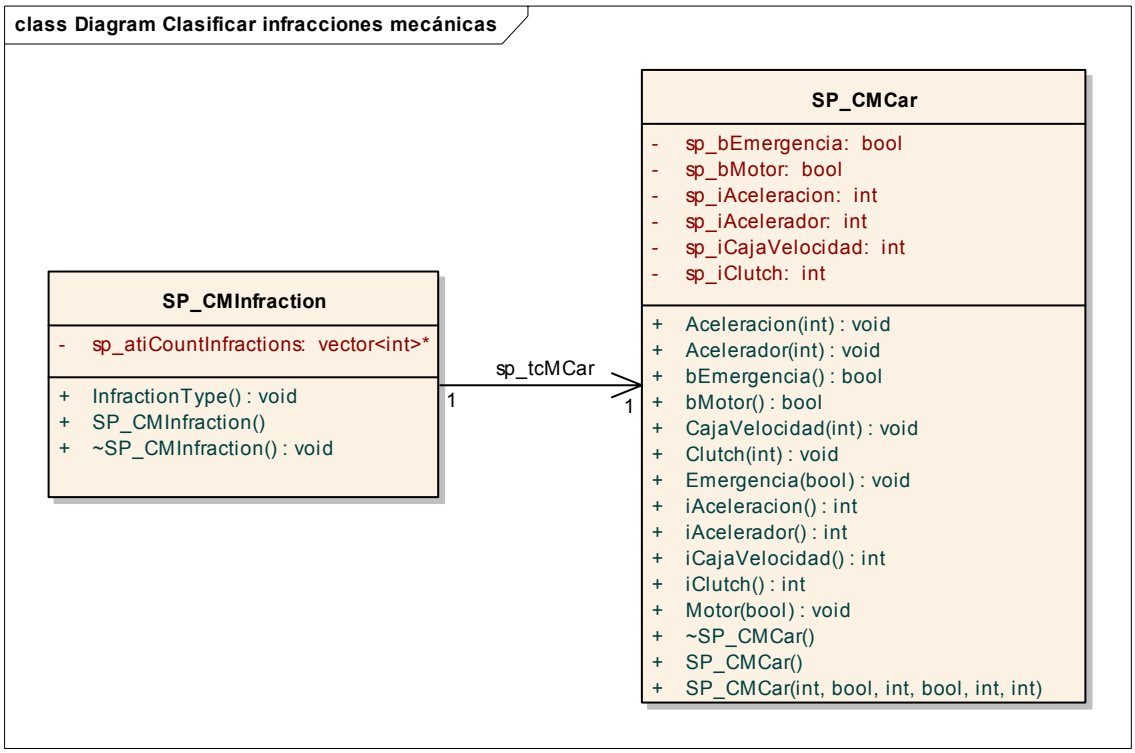

Figura 3: Diagrama de clases de diseño “Clasificar infracciones mecánicas”.

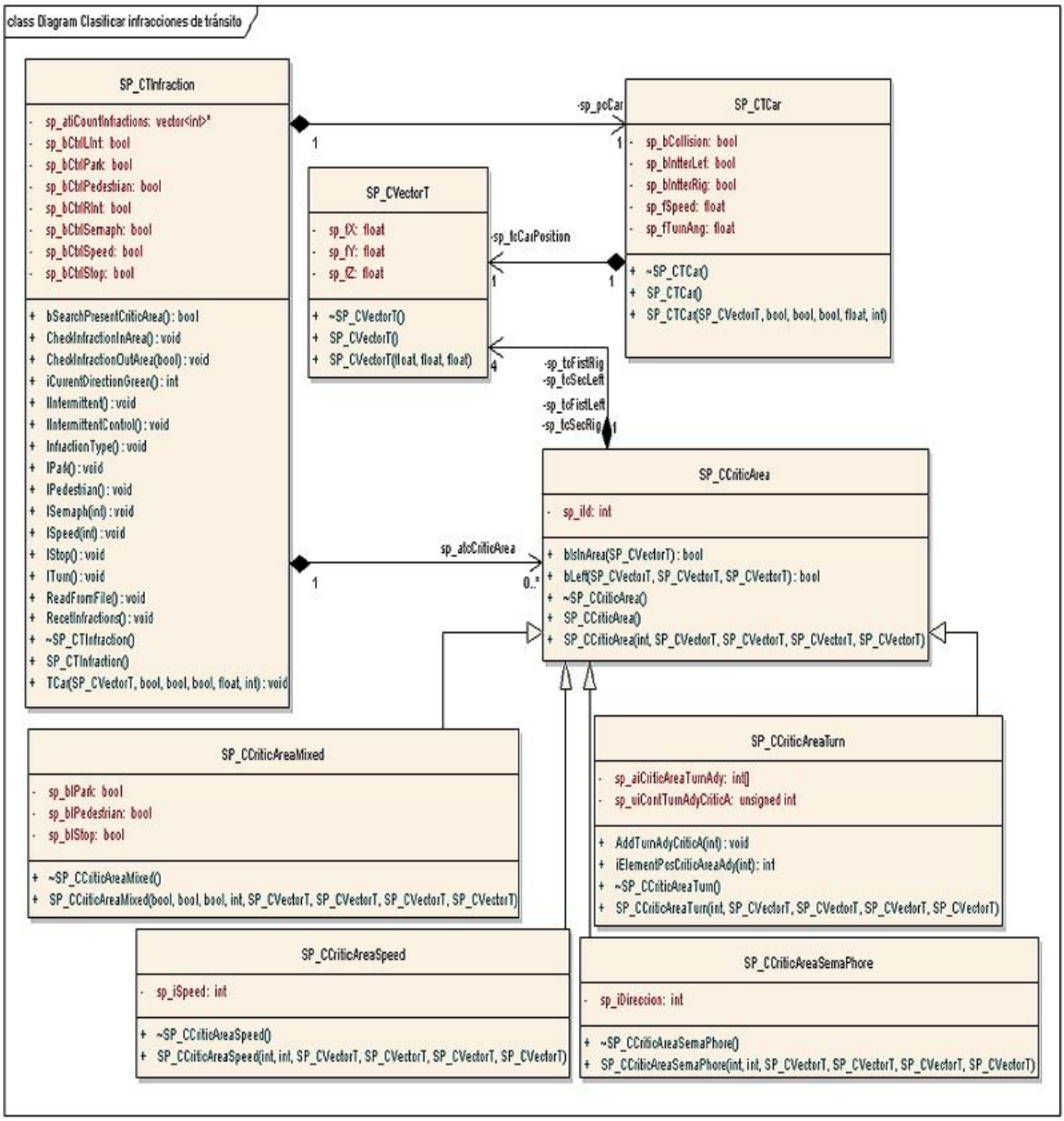

Figura 4: Diagrama de clases de diseño “Clasificar infracciones de tránsito”.

4.3 Descripción de las clases del diseño.

Tabla 8: Descripción de clase del diseño "SP_CTCar".

Nombre: SP_CTCar	
Tipo de clase: Entidad	
Atributo	**Tipo**
sp_bCollision	bool
sp_bIntterRig	bool
sp_bIntterLef	bool
sp_fTurnAng	float
sp_fSpeed	int
sp_tcCarPosition	SP_CVectorT

Tabla 9: Descripción de clase del diseño "SP_CMCar".

Nombre: SP_CMCar	
Tipo de clase: Entidad	
Atributo	**Tipo**
sp_iClutch	int
sp_bEmergencia	bool
sp_iAcelerador	int
sp_bMotor	bool
sp_iAceleracion	int
sp_iCajaVelocidad	int

Tabla 10: Descripción de clase del diseño "SP_CVectorT".

Nombre: SP_CVectorT	
Tipo de clase: Entidad	
Atributo	**Tipo**
sp_fX	float
sp_fY	float
sp_fZ	float

Tabla 11: Descripción de clase del diseño "SP_CCriticArea".

Nombre: SP_CCriticArea	
Tipo de clase: Entidad	
Atributo	**Tipo**
sp_iId	int
sp_tcFirstLeft	SP_CVectorT
sp_tcFirstRig	SP_CVectorT
sp_tcSecLeft	SP_CVectorT
sp_tcSecRig	SP_CVectorT
Para cada responsabilidad:	
Nombre:	**bLeft**(SP_CVectorT arg_tcPoint, SP_CVectorT arg_tcRectInitPoint, SP_CVectorT arg_tcRectEndPoint)
Descripción:	Determina si un punto se encuentra a la izquierda de una recta, esta función será utilizada para conocer cuando el auto esta dentro de un área.
Nombre:	**bIsInArea**(SP_CVectorT arg_tcPoint)
Descripción:	Determina si un punto se encuentra dentro de un área, será de utilidad para conocer cuando el auto ingresa en algún área.

Tabla 12: Descripción de clase del diseño "SP_CCriticAreaMixed".

Nombre: SP_CCriticAreaMixed	
Tipo de clase: Entidad	
Atributo	**Tipo**
sp_bIStop	bool
sp_bIPeatonal	bool
sp_bIPark	bool

Tabla 13: Descripción de clase del diseño "SP_CCriticAreaSemaPhore".

Nombre: SP_CCriticAreaSemaPhore	
Tipo de clase: Entidad	
Atributo	**Tipo**
sp_iDireccion	int

Tabla 14: Descripción de clase del diseño "SP_CCriticAreaSpeed".

Nombre: SP_CCriticAreaSpeed	
Tipo de clase: Entidad	
Atributo	**Tipo**
sp_iSpeed	int

Tabla 15: Descripción de clase del diseño "SP_CCriticAreaTurn".

Nombre: SP_CCriticAreaTurn	
Tipo de clase: Entidad	
Atributo	**Tipo**

sp_aiCriticAreaTurnAdy	int [4]
sp_uiContTurnAdyCriticA	unsigned int
Para cada responsabilidad:	
Nombre:	**AddTurnAdyCriticA**(int arg_iId)
Descripción:	Adiciona un nuevo identificador en la colección de identificadores.
Nombre:	**iElementPosCriticAreaAdy**(int arg_ipos)
Descripción:	Retorna el identificador de la colección de identificadores que se encuentre en la posición que recibe la función.

Tabla 16: Descripción de clase del diseño "SP_CTInfraction".

Nombre: SP_CTInfraction	
Tipo de clase: Control	
Atributo	**Tipo**
sp_atcCriticArea	vector<SP_CCriticArea*>*
sp_atcPresentCriticArea	vector<SP_CCriticArea*>*
sp_atcLastInfractionCtriticArea	vector<SP_CCriticArea*>*
sp_atcLastVisitedCtriticArea	vector<SP_CCriticArea*>*
sp_atiCountInfractions	vector<int>*
sp_pcCar	SP_CTCar*
sp_bCtrlStop	bool
sp_bCtrlSpeed	bool
sp_bCtrlPark	bool
sp_bCtrlSemaph	bool
sp_bCtrlPedestrian	bool
sp_bCtrlRInt	bool
sp_bCtrlLInt	bool

Para cada responsabilidad:	
Nombre:	**InfractionType**
Descripción:	Determina cuando y que tipo de infracciones de tránsito comete el conductor, cada vez que se cometa una infracción se incrementará el contador de infracciones que corresponde al tipo de infracción cometida.
Nombre:	**CheckInfractionInArea**
Descripción:	Determina cuando y que tipo de infracción de tránsito se comete cuando el auto se encuentra dentro de algún área, en dependencia del tipo de área que sea se realiza una llamada a las respectivas funciones que detectan las infracciones a tendiendo al tipo de área.
Nombre:	**CheckInfractionOutArea**(bool arg_bExist)
Descripción:	Determina cuando y que tipo de infracción de tránsito se comete cuando el auto se encuentra fuera de un área, ya sea cuando está en el vació o cuando cambia de área.
Nombre:	**ITurn**
Descripción:	Detecta si el conductor comete una infracción de giro.
Nombre:	**IStop**
Descripción:	Detecta si el conductor comete una infracción de pare.
Nombre:	**ISemaph**
Descripción:	Detecta si el conductor comete una infracción de semáforo.
Nombre:	**IPedestrian**
Descripción:	Detecta si el conductor comete una infracción de paso peatonal.
Nombre:	**ISpeed**
Descripción:	Detecta si el conductor comete una infracción de máxima velocidad permitida.
Nombre:	**IPark**
Descripción:	Detecta si el conductor comete una infracción de no parqueo.

Nombre:	**RecetInfractions**
Descripción:	Se encarga de restablecer en 0 las cantidades de infracciones de tránsito cometidas por el conductor.
Nombre:	**bSearchPresentCriticArea**
Descripción:	Busca las áreas donde se encuentra posicionado el auto, y a su vez inserta cada área en una colección de áreas, dicha colección es utilizada para determinar cuando se comete infracción dentro de las áreas en que se encuentre el auto.
Nombre:	**ReadFromFile**
Descripción:	Realiza la lectura de las áreas almacenadas en el fichero, cada área extraída del fichero es almacenada en una colección de áreas, que representa todas las áreas del entorno donde el conductor pude cometer algún tipo de infracción.
Nombre:	**TCar**(SP_CVectorT arg_tcCarPosition, bool arg_bCollision, bool arg_bIntterRig, bool arg_bIntterLef, float arg_fTurnAng, int arg_iSpeed)
Descripción:	Actualiza el auto de tránsito constantemente con los parámetros recibidos desde la Herramienta del simulador.

Tabla 17: Descripción de clase del diseño "SP_CMInfraction".

Nombre: SP_CMInfraction	
Tipo de clase: Control	
Atributo	**Tipo**
sp_ptcMCar	SP_CMCar*
sp_atiCountInfractions	vector<int>*
Para cada responsabilidad:	
Nombre:	**InfractionType**

Descripción:	Determina cuando y que tipo de infracciones mecánica comete el conductor, cada vez que se cometa una infracción se incrementará el contador de infracciones que corresponde al tipo de infracción cometida .
Nombre:	**MCar**(int arg_bClutch, bool arg_bEmergencia, int arg_bAcelerador,bool arg_bMotor, int arg_iAceleracion, int arg_iCajaVelocidad)
Descripción:	Actualiza el auto mecánico constantemente con los parámetros recibidos desde el Modelo matemático.
Nombre:	**RecetInfractions**
Descripción:	Se encarga de restablecer en 0 las cantidades de infracciones mecánicas cometidas por el conductor.

4.4 Diagramas de Interacción

Un diagrama de interacción modela las secuencias de intercambio de mensajes entre objetos. Un diagrama de interacción pueden representarse a través de Diagramas de Colaboración y/o Diagramas de Secuencia. El primero se organiza de acuerdo al tiempo y el último de acuerdo al espacio. [Schmuller, 2000]

El tipo de diagrama seleccionado para construir los diagramas de interacción fue el de Secuencia, debido a que muestra cómo los objetos se comunican unos con otros en una secuencia de tiempo, qué sucede en cada momento, y para ello contienen objetos con sus ciclos de vida y los mensajes que se envían entre ellos ordenados secuencialmente. [Jacobson, 2000]

4.4.1Diagramas de Secuencia

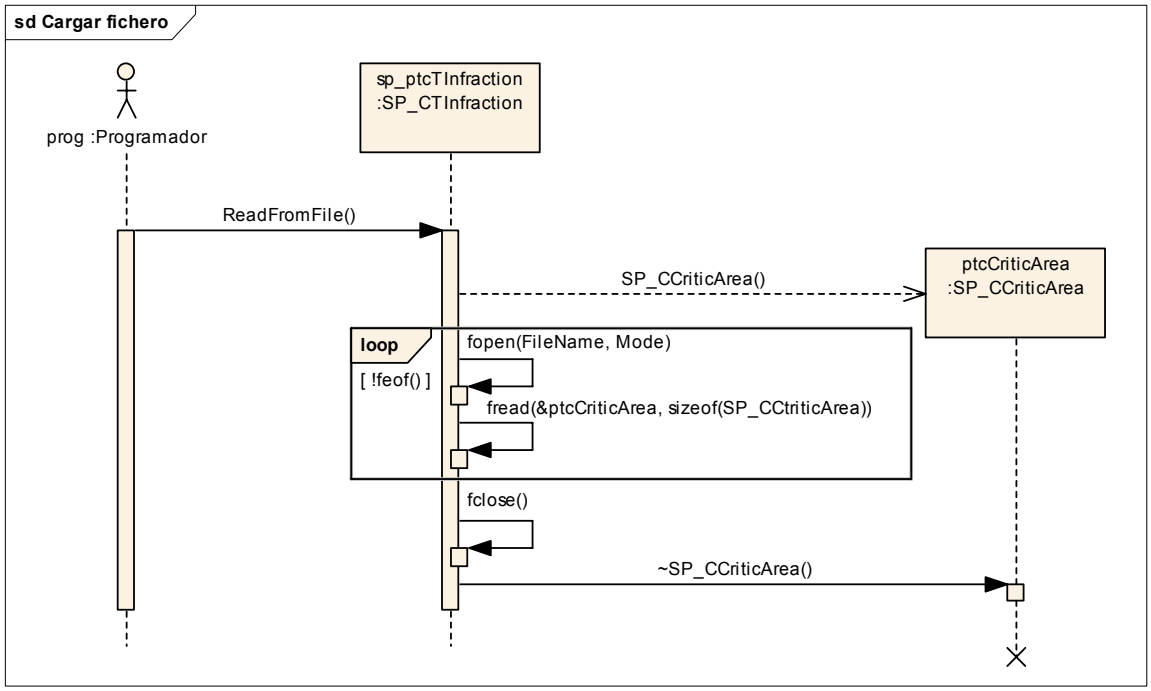

Figura 5: Diagrama de secuencia “Cargar áreas de fichero”.

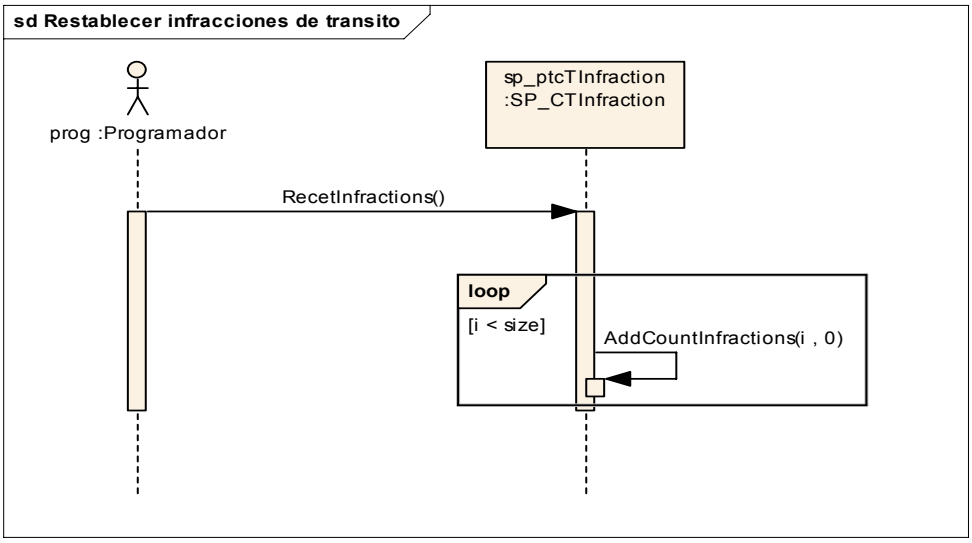

Figura 6: Diagrama de secuencia “Restablecer infracciones de tránsito”.

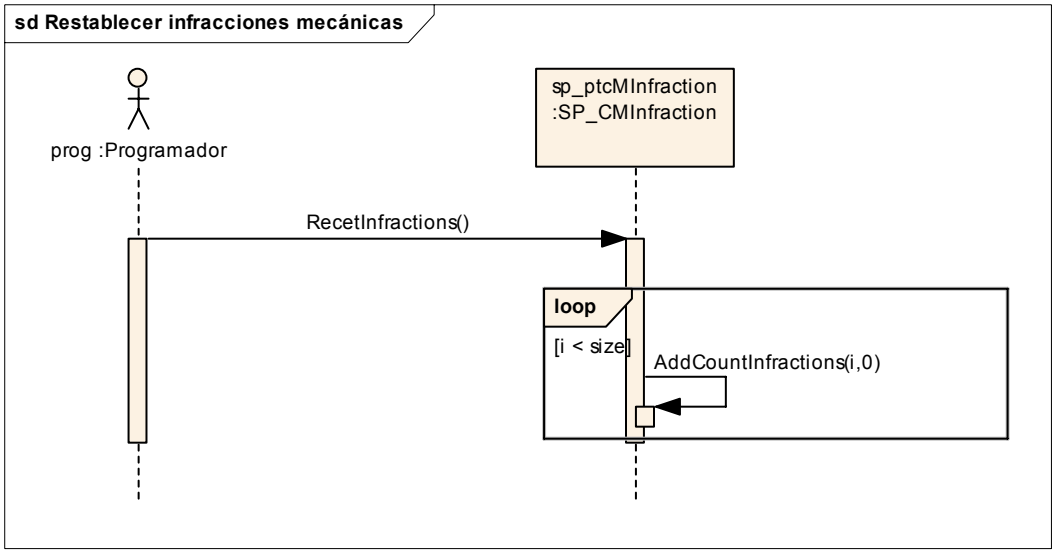

Figura 7: Diagrama de secuencia “Restablecer infracciones mecánicas”.

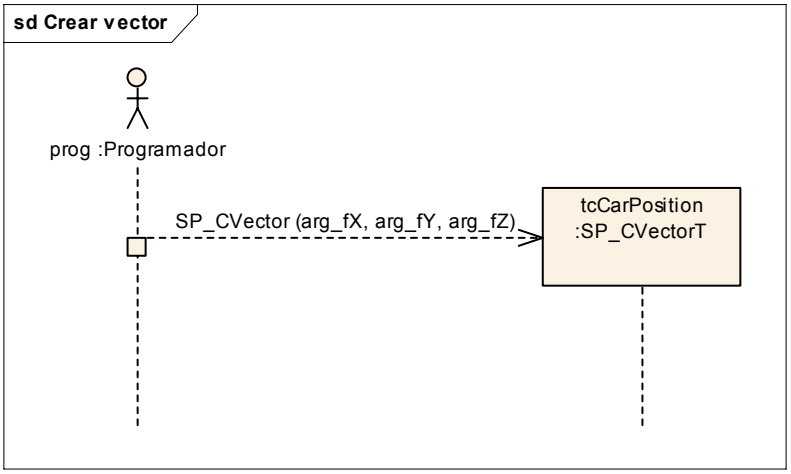

Figura 8: Diagrama de secuencia “Clasificar infracciones de tránsito”, sección “Crear Vector”.

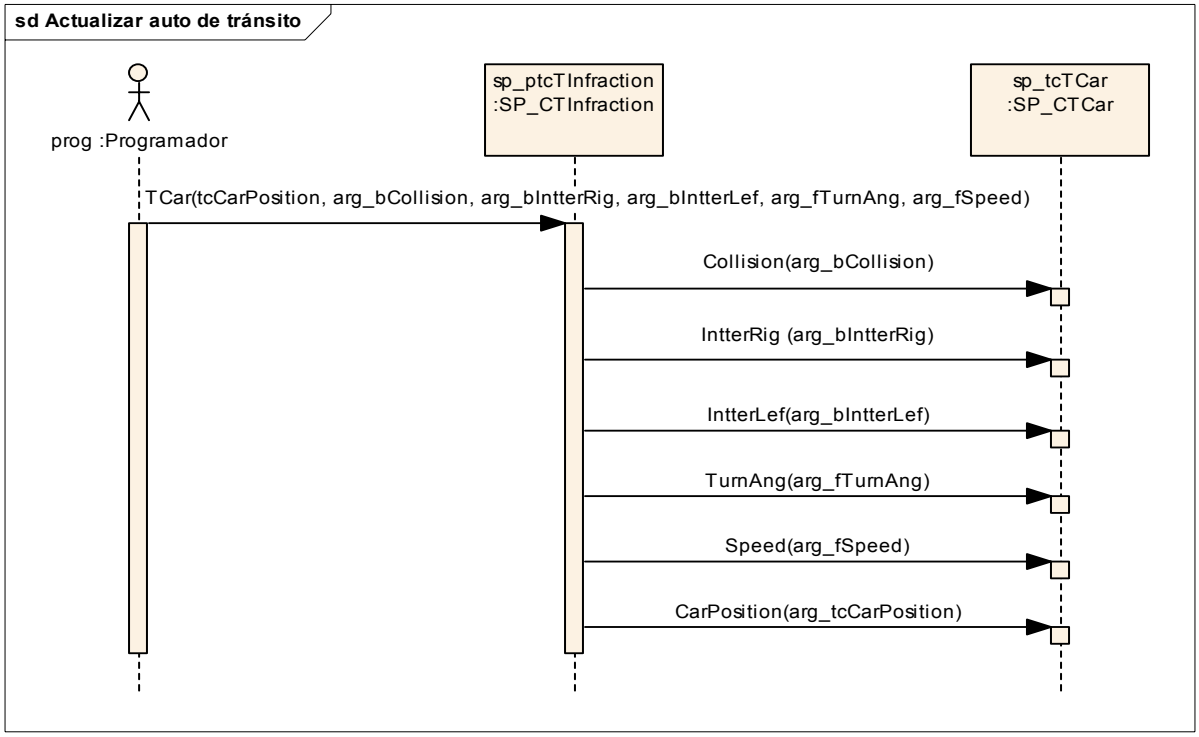

Figura 9: Diagrama de secuencia “Clasificar infracciones de tránsito” sección “Actualizar auto de tránsito”.

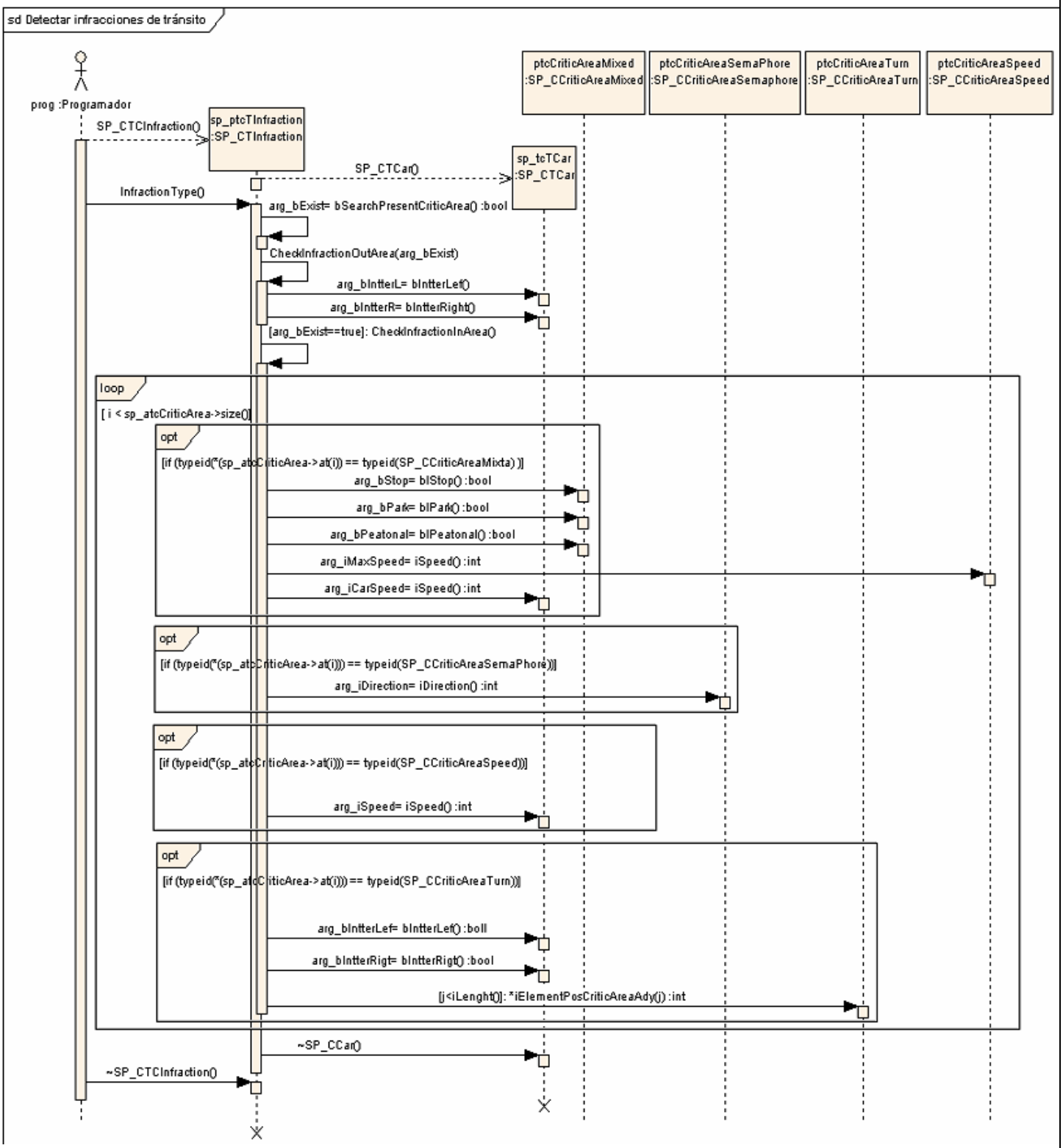

Figura 10: Diagrama de secuencia “Clasificar infracciones de tránsito”, sección “Detectar infracciones de tránsito”.

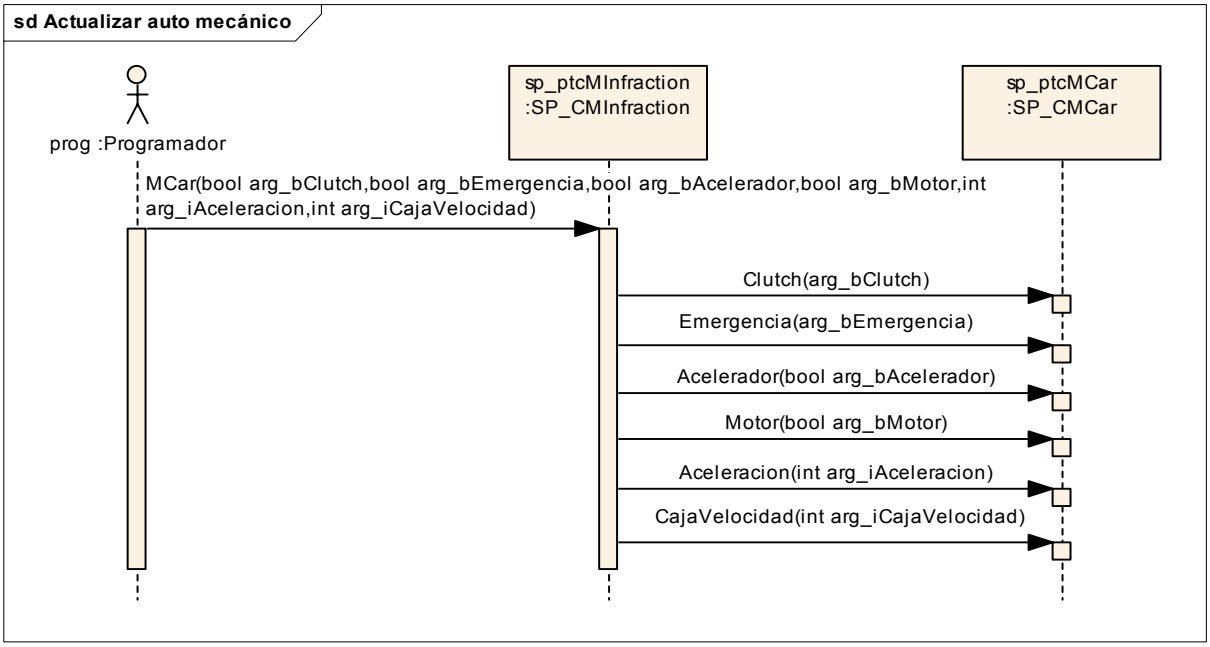

Figura 11: Diagrama de secuencia “Clasificar infracciones mecánicas” sección Actualizar auto mecánico.

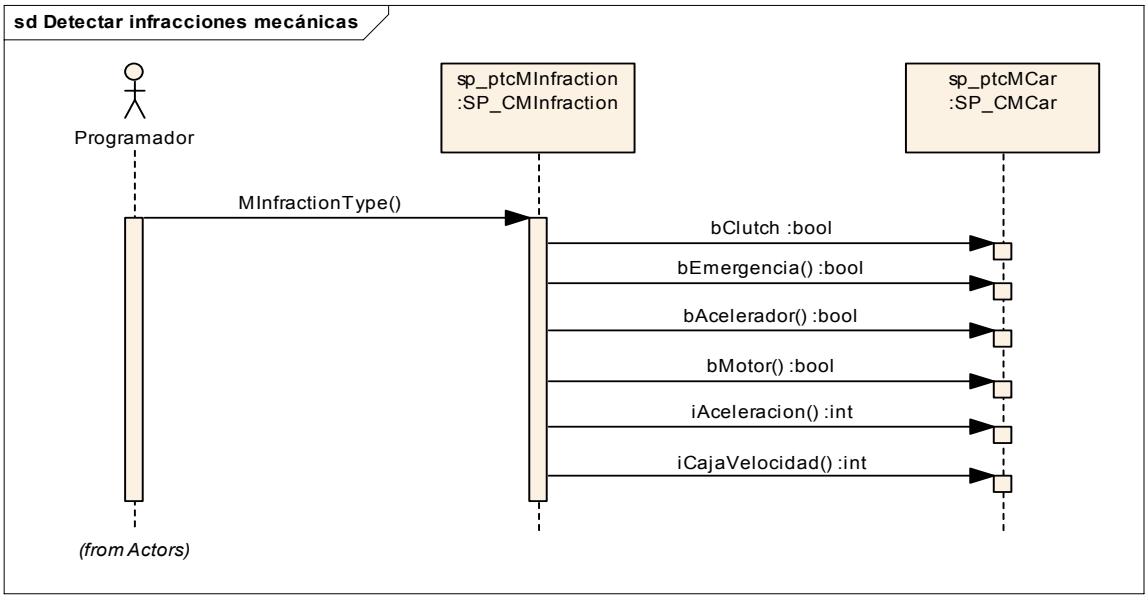

Figura 12: Diagrama de secuencia “Clasificar infracciones mecánicas”, sección “Detectar infracciones mecánicas”.

4.5 Implementación del Sistema.

4.5.1Estándares de codificación

Los estándares de codificación que a continuación se definen tienen como objetivos aumentar la portabilidad, reducir el esfuerzo de mantenimiento, y, sobre todo, mejorar la legibilidad del código desarrollado.

El idioma a utilizar para definir los nombres de variables, funciones, clases, etc., es el inglés, ya que es el idioma más difundido en el mundo informático. Se respetarán los estándares de codificación para C++.

Se nombrarán los ficheros .h y .cpp de la siguiente manera:

SPNameOfUnits.cpp

Se usará SP para identificar que pertenecen a la empresa (SIMPRO)

Tipos:

Los tipos se nombrarán siguiendo el siguiente patrón:

enum SP_EMyEnum {ME_VALUE, ME_OTHER_VALUE};

class SP_CClassName;

Las constantes se nombrarán con mayúsculas, utilizándose el "_" para separar las palabras: MY_CONST_ZERO = 0;

Las constantes de enumerados comenzarán con las iniciales del nombre del enumerado (ME_VALOR, en el ejemplo del enumerado).

Variables:

Los nombres de las variables comenzarán con un identificado del tipo de dato al que correspondan, como se muestra a continuación. En el caso de que sean variables miembros de una clase, se le antepondrá el identificador "sp_" (en minúscula), si son globales se les

antepondrá la letra “g”, y en caso de ser argumentos de algún método, se les antepondrá el prefijo “arg_”.

```
/* tipos simples */
bool bVarName;
int iName;
unsigned int  uiName;
float fName;
char cName;
char* acName;        // arreglo
char* pcName;        // puntero
char** aacName;    // bidimensional
char** apcName;    // arreglo de punteros
bool sp_bMemberVarName; //variable miembro
char gcGlobalVarName;        //variable global, no se le antepone “sp_”
/* instancias de tipos creados, se usa “t” para diferenciarlos */
SP_EMyEnumerated teName;
SP_TMyStructure ttName;
SP_AMyArray taName;
SP_CClassName tcObjectName;
SP_CClassName* ptcName;    //se lee “puntero a tipo clase” o “puntero a objeto”
SP_CClassName* atcName;  //arreglo de objetos
SP_CClassName* sp_atcName;  // variable miembro de clase
```

Funciones:

En el caso de las funciones, se les antepondrá el identificador del tipo de dato de devolución, y en caso de no tenerlo (void), no se les antepondrá nada:

```
SP_CClassName (bool arg_bVarName, float & arg_fVarName);

// solamente los constructores y destructores comenzarán con “SP_”

// en el ejemplo anterior, véase las variables argumentos

~SP_CClassName;

int* piFunction2 (…);

SP_CClassName* ptcFunction3 (…);

Procedure4 (…);
```

Se usarán los “Gets” y “Sets” para acceder a los miembros de las clases, nombrándose como los demás métodos, pero con el nombre de la variable a la que se accede:

```
char sp_cMyVar;  //variable

char cMyVar();     //”get” (sin “sp_”)

{

   return sp_cMyVar;

}

void MyVar(char* arg_cMyVar)  // “set”

{

  sp_cMyVar = arg_cMyVar;

}
```

4.5.2Diagrama de despliegue

Los Diagramas de Despliegue muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria.

Los nodos se interconectan mediante soportes bidireccionales que pueden a su vez estereotiparse. Esta vista permite determinar las consecuencias de la distribución y la asignación de recursos. [Jacobson, 2000]

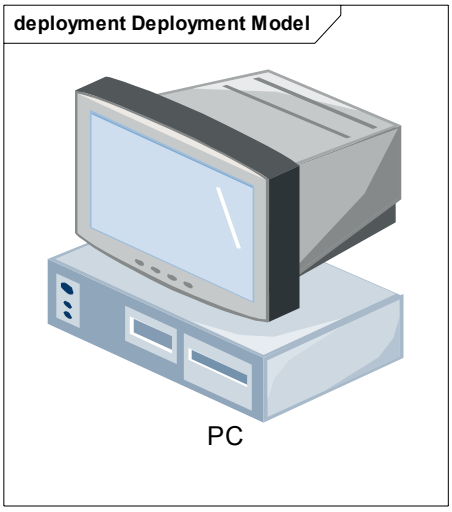

Figura 13: Diagrama de Despliegue.

4.5.3Diagramas de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Los componentes representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, ejecutables, bibliotecas cargadas dinámicamente, etc. Las relaciones de dependencia se utilizan en los diagramas de componentes

para indicar que un componente utiliza los servicios ofrecidos por otro componente. [Jacobson, 2000]

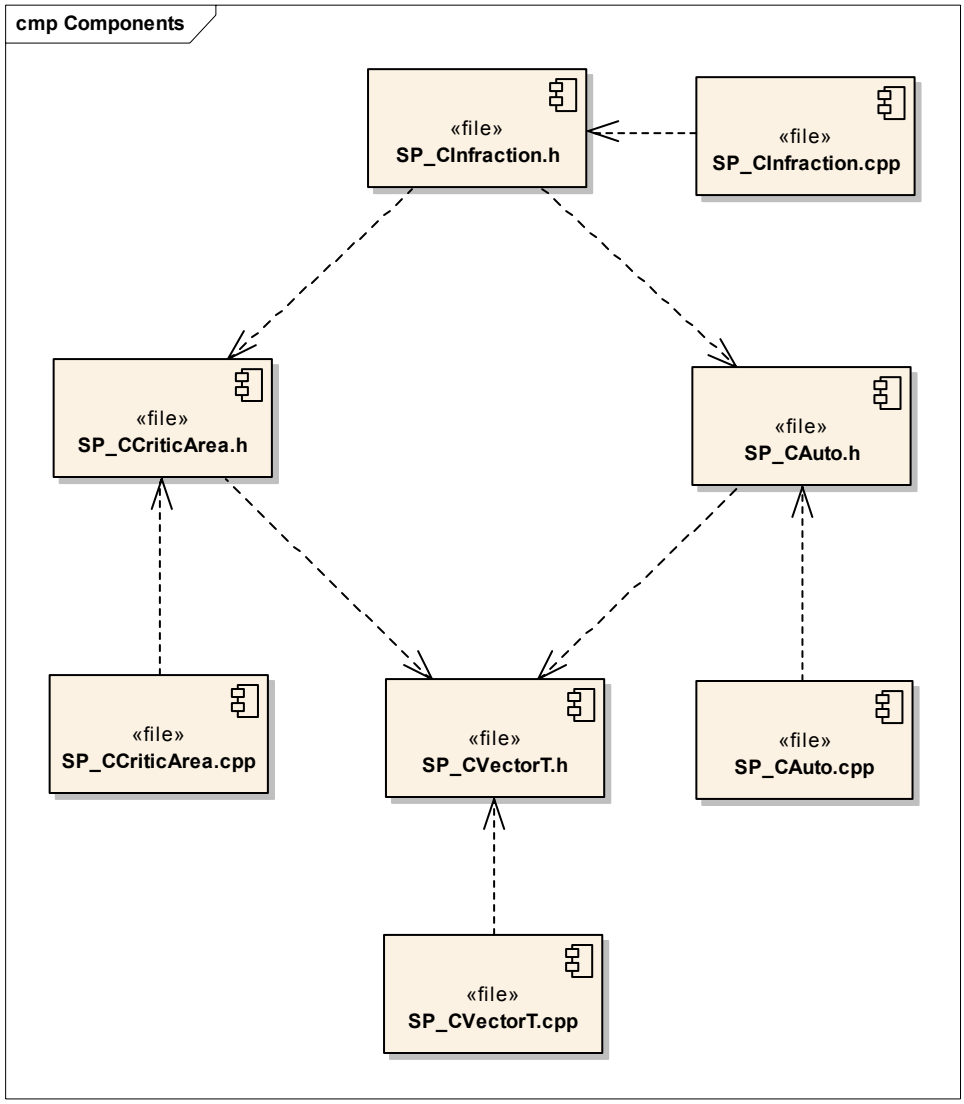

Figura 14: Diagrama de Componentes.

4.6 Descripción de los Componentes

Tabla18: Descripción de Componentes.

Componente	Contenido (Clases)
SP_CCInfraction.h SP_CCInfraction.cpp	SP_CTInfraction SP_CMInfraction
SP_CCriticArea.h SP_CCriticArea.cpp	SP_CCriticArea SP_CCriticAreaMixed SP_CCriticAreaSemaPhore SP_CCriticAreaSpeed SP_CCriticAreaTurn
SP_CAuto.h SP_CAuto.cpp	SP_CTCar SP_CMCar
SP_CVectorT.h SP_CVectorT.cpp	SP_CVectorT

4.7 Conclusiones

En este capítulo se finaliza la etapa de análisis y diseño del sistema, se definieron gráficamente los diagramas de secuencia uno por cada caso de uso, los cuales son unos de los artefactos más importantes y que requieren de mayor dedicación en esta etapa, se logró obtener un modelo de clases, donde se exponen las clases, sus relaciones y asociaciones, navegabilidad, roles y multiplicidad. Se finaliza esta etapa con el desarrollo de un diagrama de despliegue y uno de componentes, el último permitió describir los elementos físicos del sistema y la relación entre los mismos.

Conclusiones.

Con el desarrollo del presente trabajo se da cumplimiento satisfactoriamente a los objetivos propuestos. Se desarrolló un módulo que permite la detección de infracciones de tránsito del conductor en el simulador de auto SIMPRO y se planteó una propuesta con la cual se abrió camino para que otros desarrolladores e investigadores continúen con el desarrollo de la detección de infracciones mecánicas del conductor en dicho simulador. Se completó el diseño e implementación como estaba previsto. El personal involucrado en el desarrollo del presente trabajo adquirió experiencia en el conocimiento de las distintas reglas que se definieron para el manejo de las infracciones de tránsito y mecánicas cometidas por el conductor, lo cual será de gran utilidad para la futura definición de reglas para los diferentes agentes inteligentes que intervienen en el simulador.

Recomendaciones.

Tomando como base la investigación realizada y con toda la experiencia acumulada durante la realización del presente trabajo, quedan algunas recomendaciones que pueden servir de punto de partida para mejorar aún más todo lo obtenido en el presente trabajo:

- Estudiar la posibilidad de incluir otros tipos de infracciones de tránsito a medida que se vayan introduciendo señalizaciones de tránsito en el simulador.
- Estudiar para una futura implementación de las infracciones mecánicas la propuesta presentada en este trabajo, con lo cual, se les exhorta a los desarrolladores del Modelo matemático a que realicen un profundo estudio de las variables necesarias que se necesitan obtener de este Modelo para poder implementar de forma exitosa la detección de infracciones mecánicas en el simulador.
- Se les exhorta a los desarrolladores implicados en la definición de reglas de tránsito para los distintos Agentes inteligentes que intervienen en el simulador, ha que estudien, analicen y por consiguiente tomen como guía la propuesta desarrollada en el presente trabajo en la definición de reglas de tránsito para el conductor del simulador.
- Realizar la migración del Módulo a Software Libre, ya que la Herramienta del simulador SIMPRO migrará a Linux para disminuir los costos de producción y poder comercializarlo con mayores ganancias.

Referencias bibliográficas.

[Cabanellas, 2005] Cabanellas, José M. y Mera, José Manuel. *Doctorado de Tecnología de Simuladores* [en línea]. 2005. [Consultado el: 10 de diciembre 2006]. Disponible en: <http://www.gig.etsii.upm.es/jmcabanellas/Tecno_simul_0406.htm>

[Jacobson, 2000] Jacobson, Ivar; Booch, Grady y Rumbaugh, Jim. *El Proceso Unificado de desarrollo de Software* [en línea]. Addison Wesley, 2000. [Citado 5 de marzo 2007] Disponible en World Wide Web: <http://biblioteca.uci.cu/bives/titdigitales.htm>

[Larman, 1999] Larman, Craig. *UML y Patrones* [en línea]. Prentice Hall, México, 1999. [Citado 10 de marzo 2007] Disponible en World Wide Web: <http://biblioteca.uci.cu/bives/titdigitales.htm>

[Nikkel, 2002] Nikkel, Cathy. *El uso de simuladores para ayudar a entrenar a los conductores adolescentes* [en línea]. Mayo, 2002. [Consultado el: 8 de diciembre 2006]. Disponible en: <http://www.advanceautoparts.com/spanish/youcan/html/dsm/dsm20020501dt.html>

[Levis, 1997] Levis, Diego. *Que es la Realidad Virtual* [en línea]. 1997. [Consultado el: 8 de diciembre 2006]. Disponible en: <http://www.diegolevis.com.ar/secciones/Articulos/Que_es_RV.pdf>

[Schmuller, 2000] Schumller, Joseph. *Aprendiendo UML en 24 horas* [en línea]. Pearson Educación, SA. México, 2000. [Citado 5 de marzo 2007] Disponible en World Wide Web: <http://biblioteca.uci.cu/bives/titdigitales.htm>

[Steve, 2002] Steve, Rabin. *AI Game Programming Wisdom* [en línea]. Charles River Media, 2002. [Citado 21 de enero 2007] Disponible en World Wide Web: <http://www.amazon.ca/AI-Game-Programming-Wisdom-2/dp/toc/1584502894 >

Bibliografía.

[Aho, 1982] Aho, Alfred V. Hill, Murray. Hopcroft, John E. Ullman, Jeffrey D. *Data structures and algorithms* [en línea]. Addison Wesley 1982. [Citado 18 de enero 2007] Disponible en World Wide Web: <http://www.amazon.com/Structures-Algorithms-Addison-Wesley-Computer-Information/dp/0201000237>

[Boggs, 2000] Boggs Wendy, Boggs Michel. *UML with Racional Rose 2000* [en línea]. [Citado 10 de marzo 2007] Disponible en World Wide Web: <http://biblioteca.uci.cu/bives/libroddig/pdf/0782140173.pdf>

[Bourq, 2004] Bourq, David M. Seeman, Glenn. *AI for Game Developers* [en línea]. O'Reilly 2004. [Citado 19 de enero 2007] Disponible en World Wide Web: <http://www.oreilly.com/catalog/ai/>

[Camacho, 2004] Camacho Román, Yanoski Rogelio, Jiménez López, Fernando. *Biblioteca Gráfica Para Sistemas de Realidad Virtual*. Trabajo para optar por el Título de Ingeniería en Informática. La Habana. Julio de 2004.

Características de Enterprise Architect [en línea]. [Consultado el: 20 de febrero 2007]. Disponible en: <http://www.sparxsystems.com.ar/products/ea.html>

[Coca, 2006] Coca, Yuniesky. *Algunas ideas a tomar en cuenta para la inteligencia de un simulador de auto*. II taller de Realidad Virtual. Memorias del evento UCiencia 2006.

COCOMO II [en línea]. 23 septiembre 2002. [Consultado el: 6 de abril 2007]. Disponible en: <http://sunset.usc.edu/research/COCOMOII/cocomo_main.html>

Información general acerca de Visual Studio.NET 2003 [en línea]. [Consultado el: 20 de febrero 2007]. Disponible en: <http://www.microsoft.com/spanish/msdn/vstudio/productinfo/vstudio03/overview/default.asp>

Introducción a Visual SourceSafe [en línea]. [Consultado el: 20 de febrero 2007]. Disponible en: <http://msdn2.microsoft.com/es-es/library/3h0544kx(vs.80).aspx>

Simulador de vuelo [en línea]. [Consultado el: 4 de diciembre 2006]. Disponible en: <http://es.wikipedia.org/wiki/Simulador_de_vuelo>

Glosario de términos y abreviaturas.

B

Buffers: espacio de memoria para almacenamiento temporal de datos.

C

C y C++: Lenguaje de programación orientado a objetos.

CID2**:** Centro de Investigación y Desarrollo #2.

E

EA: Enterprise Architect es una herramienta de diseño y análisis UML.

Empresa SIMPRO: Empresa cubana de software" Simuladores Profesionales".

Entorno Virtual: Mundo virtual.

M

Modelo matemático: es un esquema, una ecuación, un diagrama o una teoría que simplifica una parte difícil de las matemáticas, haciendo más fácil su comprensión y que engloba de manera general muchos aspectos diferentes, es el encargado de la simulación de los elementos o sistemas incluidos en el simulador.

R

RUP: Proceso Unificado de Desarrollo de Software.

S

SIMPRO: Simulador de Conducción de Auto.

Sistemas de Realidad Virtual: sistema informático interactivo que ofrece una percepción sensorial al usuario de un mundo tridimensional sintético que suplanta al real.

SRV: Sistema de realidad virtual.

V

VSS: Microsoft Visual SourceSafe.