Simulación de transporte público: herramienta de validación y experimentación

Proyecto de Grado
Ingeniería en Computación

Fabián Coscia
Santiago Dudok
Marcelo López

Supervisores: Antonio Mauttone y María E. Urquhart

Instituto de Computación
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
Octubre de 2016

Resumen

Durante varios años el Instituto de Computación de la Facultad de Ingeniería de la Universidad de la República, a través del Departamento de Investigación Operativa, ha trabajado en el modelado de sistemas de transporte público. En ese marco se ha generado una herramienta de software que ayuda a la experimentación con dichos modelos, particularmente aquellos que utilizan la técnica de simulación. Esta herramienta está construida en base a tres componentes, a saber, uno para la creación de los modelos, otro para la simulación del modelo creado, y un visualizador de la simulación. Utilizando esta herramienta se hicieron varios estudios con el objetivo de determinar el impacto sobre los usuarios del sistema de transporte público, de las variaciones en las condiciones del servicio (recorridos, frecuencias y horarios de ómnibus) y en el comportamiento de los propios pasajeros del sistema.

Continuando con esta línea, y con el objetivo de poder modelar y experimentar con modelos que incluyen comportamientos más complejos de los pasajeros (por ejemplo, aquellos basados en información en tiempo real), este proyecto propone el desarrollo de una herramienta que permita la simulación y su visualización en forma simultánea, con el fin de poder interactuar con los modelos y validarlos también por medio de la visualización.

Dicha herramienta consta de dos partes. La primera es un proceso, que a partir de un conjunto de datos, los transforma a un formato admisible por los modelos de simulación. La segunda parte es una aplicación que simula y visualiza simultáneamente un modelo de sistema de transporte público urbano a través de los datos procesados previamente. La aplicación permite que el usuario interactúe con la simulación por medio de la visualización, por ejemplo observando la planificación del viaje de los pasajeros. Para el desarrollo de la herramienta, se implementó un modelo de simulación de base que cuenta con distintos comportamientos de pasajeros. Los criterios que pueden tomar los pasajeros para decidir su viaje, se modelan por medio de la implementación de estrategias. De esta forma, la herramienta soporta la implementación de cualquier estrategia que cumpla con las condiciones establecidas en el ciclo de vida genérico de un pasajero.

Para validar el modelo base de simulación desarrollado, se cuenta con datos de la ciudad de Rivera utilizados en proyectos anteriores, y se genera un caso de estudio para comparar los resultados obtenidos. Dicho caso se simuló de manera exitosa en la herramienta, obteniendo valores dentro del rango de validación definido.

Con el objetivo de mostrar que la herramienta soporta distintas ciudades, incluso aquellas de mediano o gran porte, se simuló un caso relativo a la ciudad de Montevideo. La simulación de este caso utilizando el modelo base se realizó con éxito, obteniéndose valores razonables en base al conocimiento de la ciudad.

Índice de contenido

Capítulo 1 - Introducción

El transporte público se ha transformado en un tema de crucial importancia en la vida cotidiana de las personas. Esto es debido a que las personas ya no se mueven dentro de un entorno acotado en una ciudad, sino que cada vez tienden a realizar viajes más largos y a diferentes destinos, haciendo que la oferta de líneas deba ser más compleja y completa.

Los sistemas de transporte público son un conjunto de elementos interactuando entre sí, para brindar el servicio de traslado de personas de un lugar a otro. Entre los elementos se encuentran la infraestructura de la ciudad, los vehículos de transporte, las normas y leyes, y los usuarios del sistema.

Los usuarios de transporte pueden comportarse de manera diferente al realizar sus viajes; por lo general tratan de minimizar el tiempo de viaje, utilizando diferentes fuentes dependiendo de la disponibilidad y fiabilidad de los diferentes tipos de información y sus preferencias.

El estudio de estos tipos de sistemas, tanto a nivel de tomadores de decisiones como de investigación, son cada vez más comunes y diversos, con el fin de identificar decisiones que permitan mejorar la relación entre la oferta y la demanda del servicio.

1.1 - Motivación

Este Proyecto de Grado de Ingeniería en Computación se enmarca en el grupo de trabajo del Departamento de Investigación Operativa (Depto. IO) del Instituto de Computación en torno al modelado de sistemas de transporte público. En proyectos anteriores, el grupo se enfocó en el estudio del comportamiento de los pasajeros al momento de realizar un viaje en un sistema de transporte público (STP), teniendo en cuenta diferentes variables y estrategias para la toma de decisiones acerca de qué línea tomar, a qué hora y en qué parada. Las decisiones que deben tomar los pasajeros son cada vez más complejas, por ejemplo están influenciadas por tecnología existente, como ser la información en tiempo real [1].

En el contexto de proyectos de investigación y desarrollo llevados a cabo por el Depto. IO para el estudio de modelos con estos comportamientos, se han desarrollado una serie de herramientas. Entre ellas, una para la generación de un modelo de un sistema de transporte público (Interfaz Gráfica para la Optimización de Recorridos de Transporte Público, IgorTP) [2]. También se cuenta con una herramienta para la simulación del modelo desarrollado. En una última etapa se desarrolló una herramienta para visualizar la salida de la simulación. La visualización consiste en un programa ejecutable, cuyos insumos son archivos de texto generados por la simulación. Por lo que es necesario que finalice la simulación para poder realizar la visualización.

Las principales limitaciones de las herramientas desarrolladas al momento son:

- no permiten interacción con el simulador en tiempo de simulación
- no es posible ejecutar la simulación y visualización en simultáneo
- complejidad al momento de ingresar o modificar datos
- degradación de performance al crecer el volumen de datos, concretamente el tamaño del modelo

Estos modelos de simulación representan la interacción detallada entre pasajeros y ómnibus, a partir de lo cual es posible calcular el nivel de servicio. Pero este depende del comportamiento de los pasajeros, que dado el abanico de opciones con que cuentan al momento de tomar decisiones, pueden ser variados y complejos. La experimentación y validación de estos modelos se ha transformado cada vez en una tarea más compleja [3]. Por lo que surge la necesidad de contar con una herramienta que permita simular y tener la visualización en tiempo real, permitiendo hacer el seguimiento de los pasajeros a medida que realizan sus viajes.

1.2 - Objetivos

El objetivo de este proyecto es modificar las herramientas existentes o desarrollar una nueva que cumpla con las características que se tienen hasta el momento, agregando las necesarias para cumplir las exigencias que motivan a este proyecto derivadas de las actividades recientes del grupo de investigación. También contar con un proceso que permita minimizar las exigencias sobre los datos de entrada de forma de ganar independencia entre el formato de los insumos y la aplicación final a desarrollar.

Además es deseable que los estudiantes adquieran conocimiento en el modelado de sistemas de transporte público utilizando simulación, entendiendo la importancia de la validación y experimentación en el contexto de un modelo de simulación. Se debe utilizar la herramienta desarrollada para validar un modelo y experimentar con el mismo.

1.3 - Resultados esperados

Diseñar una herramienta de validación y experimentación que permita la simulación y visualización en simultáneo de un modelo de sistema de transporte público. Dicha herramienta debe permitir simular diferentes casos de estudio relativos a ciudades reales, de las cuales se dispone de la información necesaria. Se plantea el caso de la ciudad de Rivera como base y el caso de Montevideo como un caso de mediano/gran porte que sería deseable lograr simular.

Se debe extender el simulador existente o desarrollar una nueva herramienta de validación y experimentación. Se desea que la herramienta permita visualizar los viajes de los pasajeros. También debe contar con un diseño tal que permita crear estrategias (comportamientos de pasajeros), y agregarlas al modelo de simulación, teniendo el menor impacto posible en el mismo. Se deben implementar al menos dos estrategias diferentes para comprobar el correcto funcionamiento de la herramienta.

Se debe validar el modelo a partir de resultados numéricos, en base a análisis de sensibilidad y comparación con resultados de trabajos previos. Para validar el simulador desarrollado se debe realizar un plan de pruebas, reportes y se debe hacer una análisis de los resultados.

1.4 - Estructura del documento

El presente documento se encuentra estructurado en siete capítulos, incluido éste. El *Capítulo 2* contiene un marco teórico para poder entender los diferentes conceptos aplicados en el proyecto.

La herramienta desarrollada es explicada en los siguientes tres capítulos. El *Capítulo 3 - Herramientas de validación y experimentación de STP*, describe las bases en las que se plantea el sistema y las funcionalidades requeridas de los diferentes componentes utilizados para la herramienta solución. Una vez descritas las funcionalidades, el *Capítulo 4 - Datos de entrada* detalla la creación del proceso para la normalización de la entrada de datos de la aplicación que modela el STP. También se explica cómo se obtuvieron los diferentes insumos del proceso para los casos de Rivera y Montevideo. En el *Capítulo 5 - Diseño e implementación de la aplicación*, primero se desarrolla cada componente en forma separada (un simulador de eventos discretos y un visualizador de la simulación a través de conceptos y tecnologías de sistemas de información geográfica), y posteriormente la integración de estos, haciendo énfasis en el comportamiento de los pasajeros a través de las estrategias.

En el *Capítulo 6 - Pruebas de la herramienta*, se describen los distintos experimentos y casos de estudio que se simularon en la herramienta. Comienza por la simulación de un caso de pequeño porte para verificar la herramienta. Luego se simulan una serie de experimentos en casos de estudio de mediano y gran porte con el objetivo de validar el modelo y la herramienta desarrollada. Para finalizar el capítulo de pruebas, se definen comportamientos específicos de pasajeros para probar funcionalidades particulares.

Finalmente, en el último capítulo, *Conclusiones y trabajo futuro*, se resumen los resultados obtenidos y las conclusiones, dejando abiertos algunos planteos para trabajos posteriores.

El documento también consta de siete anexos. En *Anexo 1*, *Anexo 2* y *Anexo 3* se presentan las decisiones de elección de herramientas y pruebas realizadas para la entrada de datos, las tecnologías de Sistemas de información geográfica (SIG) y bibliotecas de simulación, respectivamente.

En el *Anexo 4*, se detallan los datos espaciales obtenidos de la ciudad de Montevideo y un pre-proceso necesario para la adaptación de estos a la herramienta desarrollada.

El *Anexo 5*, hace referencia a los reportes generados luego que finaliza una ejecución de la aplicación, describiendo los diferentes campos configurados y su utilidad para posibles análisis a realizar.

Por último, los *Anexo 6* y *Anexo 7* son los manuales de usuario y de desarrolladores, respectivamente. Con ellos se realiza una descripción de las principales funciones de la herramienta, la estructura del desarrollo y de sus posibles modificaciones.

Capítulo 2 - Marco teórico

En este capítulo se exponen conceptos relativos a las áreas que involucra el proyecto con el fin de su compresión. El abordaje es por área de estudio, yendo a lo particular tanto como se requiera para satisfacer las referencias del presente informe. Se han distinguido tres áreas de estudio: sistemas de transporte público, simulación a eventos discretos y sistemas de información geográfica. A continuación se expone cada una de ellas.

2.1 - Sistemas de transporte público - STP

En esta sección se definen conceptos de los sistemas de transporte público, en particular los referidos al transporte público urbano, y una visión general de su modelado [4].

Un sistema de transporte público urbano se puede definir como un sistema de vehículos e infraestructuras de transporte colectivo de pasajeros, que permite el desplazamiento de personas de un punto a otro dentro de una ciudad o región bien definida. La oferta está definida por los servicios (que son efectuados por vehículos que realizan diferentes recorridos con determinada frecuencia), mientras que la demanda está definida por los usuarios y sus requerimientos de movilidad entre diferentes puntos de la ciudad.

Un STP está compuesto por diversos componentes que interactúan entre sí y que se pueden categorizar de la siguiente manera:

- Infraestructura: calles, carreteras, etc.
- Vehículo: automóvil, ómnibus, etc.
- Operador: persona o sistema que guía un vehículo.
- Usuarios: personas.

Dentro de un STP, se distinguen tres actores

1. Las entidades reguladoras del sistema que se encargan de dictaminar las paradas, los recorridos y las frecuencias de cada línea de ómnibus.
2. Las empresas de transporte ofrecen el servicio a través de la flota de ómnibus.
3. Los usuarios, pasajeros de los ómnibus que realizan viajes de un origen a un destino.

Además, un sistema de trasporte publico consta de un proceso de planificación de tránsito, comúnmente incluye cuatro actividades básicas que por lo general se realizan en secuencia [5]:

1. El diseño de rutas.
2. Planificación de frecuencias de lineas y tabla de horarios.
3. Asignación de flota.
4. Asignación de personal y recursos disponibles para los viajes

Un pasajero puede realizar su viaje de forma directa o a través de transbordos. Se define viaje directo cuando un pasajero realiza el viaje utilizando solo un vehículo, y transbordo cuando utiliza más de un vehículo.

2.1.1 - Modelado de STP

En términos generales, es costoso e inapropiado experimentar con un STP real, por lo que es necesario realizar un modelo que permita el estudio y que sea lo más representativo posible de la realidad [6][7]. A continuación se describe cómo se modela un STP en el contexto de este proyecto.

Modelado de la infraestructura y demanda

Una ciudad se divide en regiones a las que llamaremos zonas o zonas de demanda. Cada zona cuenta con un único punto de concentración de la demanda, al que llamaremos centroide. Se asume que todas las personas que parten o arriban desde o hacia una zona, lo hacen a través del centroide. La demanda representa la frecuencia de viajes requeridos por los pasajeros entre distintas zonas. Esta representación se realiza

mediante una matriz origen-destino (OD), que expresa las tasas de viajes individuales de un centroide origen hasta un centroide destino. Cada elemento de esta matriz se llama par-OD.

La red de calles de la ciudad se modela mediante un grafo. Dicha red está compuesta por segmentos o aristas que pueden tener una ponderación (como por ejemplo tiempo medio de viaje del segmento) y nodos que representan a los centroides, los cruces de la red vial y las paradas de ómnibus. Las paradas de ómnibus se conectan a través de aristas (denominadas caminatas) al centroide de la zona en que se encuentran. También se pueden modelar caminatas entre dos paradas de una misma zona.

En la Figura 1 se muestra un ejemplo que ilustra cómo es el modelado de la infraestructura y la demanda.

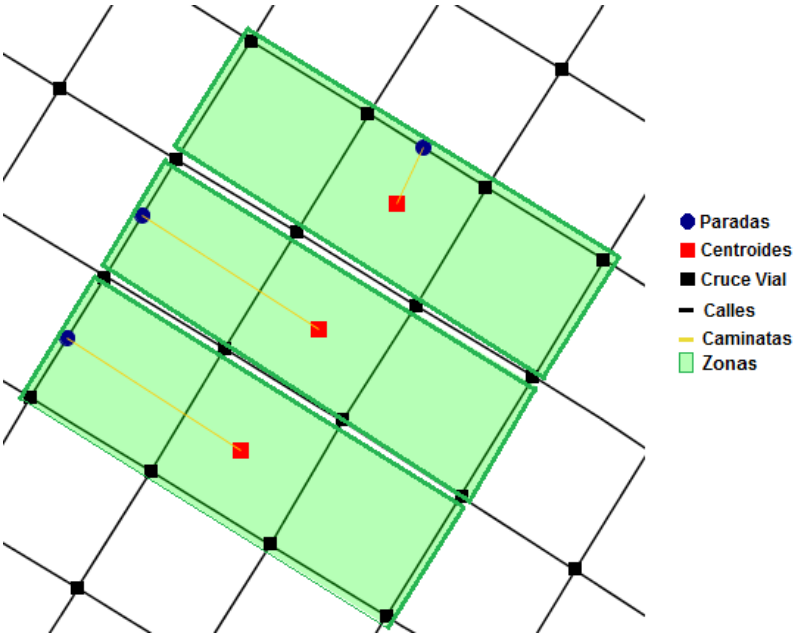


Figura 1 - Representación del modelado de Infraestructura y demanda

Modelado de líneas y recorridos de ómnibus

La líneas de ómnibus se modelan como una entidad que contiene un recorrido circular o dos recorridos (Ida y Vuelta), y un calendario para la salida de ómnibus. Los recorridos se modelan como una secuencia ordenada de nodos (cruces y paradas) y las aristas que unen a estos nodos. El calendario de salida puede ser de dos maneras: una tabla de horarios, que informa el tiempo de salida de cada ómnibus para la línea o una frecuencia de salida, que informa cada cuanto sale un ómnibus de la línea.

En la Figura 2 se muestra la representación gráfica del modelado de un recorrido de una línea.

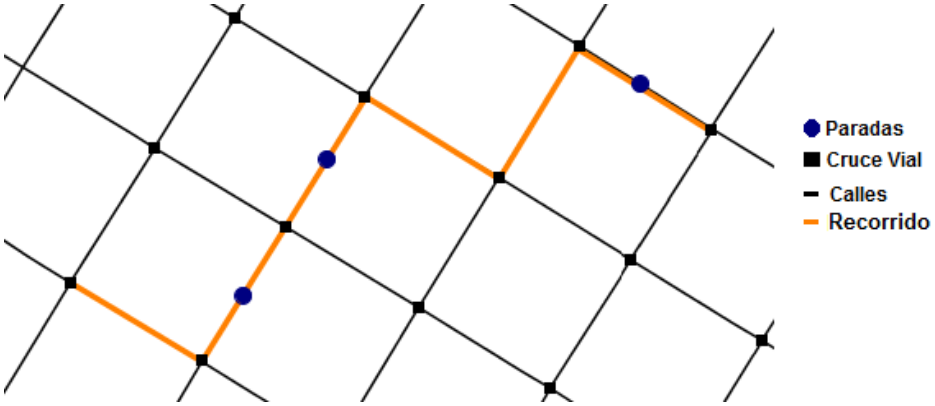


Figura 2 - Representación del modelado de una línea en un grafo

Modelado del comportamiento de los pasajeros

En cuanto a los pasajeros y su comportamiento es donde recae la mayor atención en este proyecto. El modelado del comportamiento de los pasajeros -denominado estrategia- se define por un conjunto de reglas que aplica al pasajero para llegar a su destino.

Los pasajeros siempre toman un criterio (como por ejemplo minimizar una medida, o por el contrario, maximizar la utilidad de otra), por lo que deben elegir el camino para viajar desde el origen al destino utilizando un determinado conjunto de recorridos de líneas de ómnibus. En este contexto, las decisiones pueden referirse a la hora de salida desde el centroide origen, qué parada de ómnibus es la inicial, la línea (o un conjunto de líneas) a tomar, en qué parada debe bajar para cada línea.

Modelado de la aleatoriedad

Con el fin de modelar las variaciones en el tiempo de viaje de los ómnibus (por ejemplo debido a diferentes factores, como las características del conductor o las condiciones del tráfico) y/o modelar variaciones de frecuencia en la creación de pasajeros para ir de un origen a un destino, se deben usar distribuciones de probabilidad. Estas son importantes fuentes de aleatoriedad de un modelo, y son características que afectan el comportamiento de pasajeros y el rendimiento del sistema bajo diferentes condiciones.

2.2 - Simulación a eventos discretos - SED

La simulación de sistemas generalmente se refiere a la construcción de un modelo abstracto que representa algún sistema de la vida real. Describe los aspectos pertinentes del sistema como una serie de ecuaciones, relaciones, y/o sentencias lógicas embebidas en un programa de computación.

Los conceptos vertidos en esta sección fueron obtenidos del libro *Simulation Modelling with Pascal*, en que se basa el curso de Simulación a Eventos Discretos dictado por el Inco [8].

La simulación a eventos discretos (SED) modela un sistema como una secuencia discreta de eventos en el tiempo. Cada evento tiene lugar en un momento determinado en el tiempo y marca un cambio de estado en el sistema. Entre los eventos consecutivos, se supone que no se produce ningún cambio en el sistema. Así, la simulación puede saltar directamente en el tiempo de un evento a otro. Esto contrasta con la simulación continua, en la que se desea representar cada cambio del sistema conforme avanza el tiempo.

Al momento de modelar un sistema a simular se debe especificar claramente:

- Objetivos del estudio: en base al propósito de la simulación. De esta forma se debe seleccionar qué elementos se modelan efectivamente y cuáles pueden ser obviados.
- Hipótesis iniciales: que se asumen respecto al sistema real y su correspondiente modelo.
- Variables de decisión: elementos bajo control del tomador de decisiones, influyen en las salidas del modelo, por ejemplo, medidas que representan la performance del sistema real.
- Respuestas: variables que responden al objetivo del estudio.

Los propósitos de realizar una simulación pueden ser:

- De comparación: dos ejecuciones de una simulación pueden ayudar a ver el efecto de cambiar una variable de decisión.
- De predicción: una simulación puede ser usada para predecir el estado de un sistema en un instante de tiempo.
- De investigación: algunas simulaciones pueden ser construidas de forma de proveer un pantallazo del sistema en lugar de realizar una experimentación detallada.

Los objetos o individuos cuyas actividades se quieren modelar son representados por entidades. Los objetos que no tienen atributos pero actúan como restricciones a las actividades de las entidades son llamados recursos. Según la metodología de SED, los cambios de estado en las entidades ocurren en puntos discre-

tos del tiempo, que son llamados eventos. En la metodología introducida por Tocher, hay dos tipos de eventos (Capítulo 2 de *Simulation Modelling with Pascal*) [8]:

- Evento agendado (B event, bound) o fijo: su ocurrencia es predecible y puede ser agendado.
- Evento condicional (C event): su ocurrencia depende de ciertas condiciones del sistema.

Los estados de una entidad (una vez creada) pueden ser:

- Ocupada, cuando está agendada para algún evento fijo.
- En cola, en espera del turno para que alguna condición sea satisfecha.
- Desocupada, inactiva u ociosa, entidades que no están ni ocupadas ni en cola.

Generalmente en SED interesa saber el tiempo insumido por las entidades, el cual generalmente se da en colas o en actividades.

El ejecutivo o mecanismo de avance del tiempo es el procedimiento (programa) que se encarga de que los eventos sucedan en el orden correcto. Se estructura según tres métodos de estructuración de la simulación o "enfoques del mundo":

1. Tres fases: orientado a eventos, donde los eventos condicionados y fijos se programan como procedimientos separados. Las tres fases son: Avance del reloj, ejecución de eventos fijos, ejecución de eventos condicionados.
2. Dos fases: orientado a eventos, donde los procedimientos que describen los eventos fijos incluyen todos lo eventos condicionados que van a suceder como resultado de los propios eventos agendados o fijos. Las dos fases son: Avance del reloj, ejecución de eventos fijos.
3. A procesos: orientado a procesos, las acciones realizadas por cada clase de entidades se proyectan como un proceso, el cual describe el ciclo de vida de la entidad.

Para simular el comportamiento del sistema se utiliza valores de variables aleatorias, que deben ser muestreados. El uso de variables aleatorias aporta realismo al modelo de simulación. Usar valores determinísticos en general no es adecuado, ya que puede generar resultados sesgados, poco realistas. Por lo tanto, se deben utilizar distribuciones de probabilidad, cuyas muestras son generadas mediante números aleatorios. Como es costoso obtener números realmente aleatorios, en computación se utilizan generadores de números pseudo-aleatorios. Estos son números generados en un proceso que parece producir números al azar, pero no lo hace realmente. Son predecibles a partir del primer número denominado semilla. Las distribuciones utilizadas para obtener los valores de variables aleatorias en el proyecto son: Distribución Uniforme, Distribución Normal, Distribución Exponencial Negativa, Distribución Discreta General (Capítulo 4 de *Simulation Modelling with Pascal*) [8].

La simulación comienza en el tiempo cero y ejecuta todos los eventos en el orden en el cual ocurren, hasta que se cumple alguna de las siguientes condiciones:

- No hay más eventos que ejecutar.
- El tiempo de ejecución del próximo evento supera el máximo estipulado para la simulación.
- Se ejecutó un evento que pone fin a la simulación.

Si se desea observar el comportamiento general de la simulación alcanza con realizar una sola corrida. En cambio, si se desea obtener medidas o valores estadísticamente válidos deben realizarse varias corridas con distintos números aleatorios que generarán distintos valores de las distribuciones muestreadas.

Se obtienen datos durante las corridas para realizar los cálculos necesarios que fueran determinados según los objetivos de la simulación. Estos deben ser producidos en cantidades digeribles de modo que puedan ser resumidos en histogramas (representación gráfica usada para mostrar resultados de la simulación o la tendencia de algunas variables durante la simulación), tablas y/o reportes.

La salida visual de una SED es utilizada para diferentes motivos, como lo pueden ser para validación del modelo, realizar un análisis visual o comunicarle a los usuarios algunos aspectos de la simulación.

Entre los métodos para realizar la salida visual se encuentran:

- Imprimir variables de la simulación.
- Mostrar los histogramas a medida que se ejecuta la simulación.
- Diagramas del estado del sistema a través de sus entidades y recursos.
- Representación gráfica de la simulación.

La representación gráfica de la simulación puede ser mediante una codificación explícita de los componentes, o mediante el registro de información para un visualizador. En este tipo de representación por lo general hay componentes estáticos, que se encuentran fijos durante toda la simulación, y componentes dinámicos que cambian su estado a través de los eventos.

Todo modelo debe ser testeado para asegurarnos que es confiable, no tiene errores y que es aceptado por aquellos que lo van a usar. La verificación es la tarea de chequear el modelo y el programa para asegurarnos de que se comporte como esperamos. Luego de realizada la verificación, se trata de determinar el comportamiento del modelo bajo diversas circunstancias, variando sus factores (en particular, las variables de decisión) para estimar los parámetros de interés. Este proceso se llama experimentación.

En cualquier trabajo de modelado -y muy especialmente en simulación-, el proceso de plantear hipótesis, construir el modelo y validarlo es un proceso cíclico. Muchas veces las partes individuales de un modelo parecen representar la realidad, pero cuando se consideran en conjunto, no resulta un reflejo acorde de la conducta del sistema en general. Existen dos formas de validar un modelo:

- Permitir que el usuario chequee que la simulación se desarrolla como debe. El usuario debe poder entender el diagrama de actividades y debe participar activamente en el planteo de los objetivos del trabajo, y por ende en la lógica y detalles de la simulación. Es importante brindarle resultados visuales del comportamiento de las colas, entidades y el uso de recursos, que le permitan ver si la simulación se comporta en forma similar al sistema real.
- Brindar estadísticas que confirmen que la simulación produce resultados similares a los del sistema real. Esto necesita de una recolección de datos adicional acerca de promedios de largos de colas, distintos tiempos, los que se confrontan con los obtenidos mediante la simulación.

2.3 - Sistemas de información geográfica - SIG

Un Sistema de Información Geográfica (SIG), es la combinación de cinco componentes: personas especializadas, datos descriptivos y espaciales, métodos analíticos, hardware y software; organizados para analizar, manipular, procesar, almacenar, generar y visualizar todo tipo de información referenciada geográficamente [9]. Sin faltar a la definición, en muchos pasajes de este documento se llama SIG a un componente del sistema solución, justamente por entender que está enmarcado en el concepto anterior. Además de que este provee funcionalidades típicas que caracterizan a dichos sistemas de información.

Una de las principales motivaciones para la utilización de un SIG, es su prestación de gestión de información geoespacial y su carácter casi inherente de visualización. Este tipo de sistemas permite separar la información en diferentes capas temáticas y las almacena de forma independiente. Esto facilita el trabajar con ellas, además de permitir relacionar la información existente, a través de la topología geoespacial de los objetos.

Como se desprende de la definición, los datos son un punto crucial para este tipo de sistemas. Por el carácter introductorio de este capítulo, no se es exhaustivo en las distintas variantes de datos, tipos, características y otros, sino que se ciñe a los puntos que se relacionan al presente proyecto y en concreto a su solución. Un tipo de dato que es el utilizado en este proyecto es el Vectorial: proporciona una manera de representar objetos espaciales del mundo real dentro de un ambiente SIG [10]. Un objeto espacial puede ser un ómnibus, una manzana, una calle, etc. Los objetos espaciales también se los denomina como entidades o features.

El interés de la representación, se centra en la ubicación de los elementos geográficos sobre el espacio, y donde los fenómenos a representar son discretos, es decir, de límites bien definidos. Un objeto espacial obtiene la forma en la representación utilizando su geometría. La geometría se compone de uno o más vértices interconectados. Un vértice describe una posición en el espacio utilizando una coordenada <X,Y>. Cuando la geometría de un objeto espacial está compuesta de un solo vértice, se conoce como una entidad del tipo punto. Cuando la geometría consiste en dos o más vértices y el primer y último no son iguales es un elemento del tipo línea. Cuando tres o más vértices están presentes, y el último coincide con el primero, se forma un objeto espacial del tipo polígono. Para elementos más complejos que puntos, el orden de los vértices puede interpretarse como la definición de un sentido, siendo el primer vértice el origen, y el último el fin. Esta observación es conveniente en ciertos contextos, en otros hasta carece de sentido según el tipo del objeto espacial. Cuando varias entidades de tipo línea representan un objeto espacial se dice que es de tipo polilínea. Por ejemplo, se puede representar el recorrido de un ómnibus como una polilínea compuesta por la unión de varios tramos donde cada uno de ellos es una entidad de tipo línea.

El almacenamiento de la información de los SIG, se hace a través de bases de datos. A cada capa le corresponde una tabla de la base. Cada tabla es definida por un esquema que declara los atributos que posee, y de qué tipo son. Por lo que todas las entidades de una misma capa son del mismo tipo. Un SIG puede albergar varias capas de distintos tipos. Cada entidad está vinculada a una fila de una tabla, donde hay un atributo que almacena su geometría. Además, en general, poseen otros atributos que pueden ser texto, información numérica o booleana que describe los objetos espaciales.

Un Shapefile es un formato de archivo propietario de datos espaciales del tipo vectorial, desarrollado por la compañía ESRI [11]. Originalmente se implementó para sus productos, pero actualmente se ha convertido en un estándar de facto. Es un formato multiarchivo, esto implica que está compuesto por varios archivos. La extensión de los archivos mínimos para configurar correctamente un shapefile son los siguientes:

- .shp: es el archivo que almacena las entidades, tanto la geometría como sus eventuales atributos.
- .shx: es el archivo que almacena el índice de las entidades.
- .dbf: es la base de datos (con una sola tabla), donde se almacena la información de los atributos de los objetos.

Por simplicidad, de aquí en más cuando se hable de archivos shape, shapefiles o “nombre_archivo.shp” se estará haciendo referencia a el conjunto anterior.

Otro concepto importante es el de proyecciones cartográficas o geográficas: tienen como objetivo el representar y/o visualizar la superficie curva de la Tierra o una parte, en una superficie plana como un mapa. Un sistema de coordenadas de referencia (CRS sus siglas inglés) define cómo el mapa bidimensional proyectado en un SIG, es relacionado con superficies reales del planeta. La decisión sobre el sistema de proyección cartográfica y el sistema de coordenadas de referencia a usar, depende de la extensión regional en que se desea trabajar, del trabajo en sí, y a menudo de la disponibilidad de datos [12]. Los sistemas de coordenadas tienen un código que los identifica, a través del cual podemos conocer los parámetros asociados al mismo, se trata del SRID. Para representaciones de Uruguay, es típico usar el CRS: WGS 84 / UTM zone 21S y SRID: EPSG: 32721.

Capítulo 3 - Herramienta de validación y experimentación de STP

En este capítulo se describen las funcionalidades de la herramienta solución de este proyecto. Si bien hubo ciertos requerimientos iniciales, estos se fueron refinando durante el transcurso del proyecto, resultando en las funcionalidades finalmente implementadas. La solución consta de dos partes. La primera es un proceso, que a partir de un conjunto de datos, los transforma a un formato definido. Dicho proceso, como su fin así lo define, se llamará transformación de datos. La segunda parte es una aplicación que simula y visualiza simultáneamente un modelo de sistema de transporte público urbano a través de los datos procesados.

Siguiendo el orden temporal del uso de la herramienta, se aborda en la sección *3.1* el proceso de transformación de datos, particularidades y aspectos que fundamentan su existencia. Luego, en la sección *3.2*, la atención está puesta en la aplicación, estableciendo las características que hacen al modelado del presente proyecto de un sistema de transporte público urbano, haciendo principal hincapié en el modelado del comportamiento de sus pasajeros. Si bien en este capítulo se describen los requerimientos y funcionalidades, dado que estos implican cuestiones de modelado, se dan detalles al respecto y se especifican en profundidad en el *Capítulo 5 - Diseño e implementación de la aplicación.*

3.1 - Proceso de transformación de datos

La transformación de datos permite modificar datos no válidos para la entrada de la aplicación -pero con algunas características particulares- a datos válidos. La aplicación exige datos con atributos determinados, como varios tipos de identificadores y relaciones entre estos. El conocimiento del desarrollo de la aplicación, permite fabricar sobre datos de entrada no válidos los atributos y valores que deben poseer, así como construir relaciones para que se conviertan en datos válidos, en el sentido de satisfacer los requerimientos de la aplicación. Es necesario aclarar que no todo juego de datos no válido que atraviese el proceso se transforma en uno válido, claramente deben cumplir una serie de exigencias que se detallan en *Capítulo 4 - Datos de Entrada*.

Este proceso es una pieza fundamental de la solución del proyecto, ya que amplía el espectro de datos candidatos a estudiar con la aplicación. Por otro lado, si los datos disponibles respetan los requerimientos de la aplicación, no es necesaria su transformación para ser utilizados en el simulador.

Como base del estudio del presente proyecto se utiliza los datos disponibles de la ciudad de Rivera, Uruguay. Estos datos están disponibles de proyectos anteriores. En particular su formato se definió para la herramienta IgorTP [2]. Las características de estos datos no son excluyentes, pero ayudan a establecer exigencias mínimas que debe cumplir cualquier juego de datos de entrada.

El proceso presenta tres características: una entrada bien definida, el proceso en sí, y una salida resultante de la ejecución del mismo. La entrada está dada por un conjunto de shapefiles: de paradas, de líneas y de zonas. Estos archivos deben respetar ciertas características particulares. El proceso consiste en la ejecución de componentes que se detallan en el *Capítulo 4 - Datos de Entrada*. La salida es un conjunto de archivos shape, resultante de la ejecución del proceso. Algunos son modificaciones de la entrada y otros son copias nuevas de la entrada con modificaciones pertinentes. Las especificaciones precisas, su interpretación y su uso también se detallan en *Capítulo 4 - Datos de Entrada*.

3.2 - Funcionalidades de la aplicación

En esta sección se explica el funcionamiento de la aplicación desarrollada. En las primeras tres subsecciones se detalla el modelado del STP. En un principio se describe el ciclo de vida de todo pasajero. En segunda instancia, se detalla el modelado del STP con metodologías SED. La tercer subsección se aboca al comportamiento de los pasajeros, explicando las estrategias propuestas para este proyecto y cómo crear nuevas estrategias. Por último se realiza una descripción de la visualización, como se integra con el simulador, y sus funcionalidades de interacción con los usuarios.

3.2.1 - Ciclo de vida del pasajero

A continuación se muestra el ciclo de vida de los pasajeros que soporta el sistema desarrollado. Este es la base para realizar el diseño del sistema. Tanto la simulación como la visualización están estrechamente vinculadas a este flujo, por lo que cualquier estrategia que realice un pasajero tiene que cumplir con este ciclo. En la Figura 3 se muestra el flujo correspondiente al ciclo de vida.

1. Creación de un pasajero.
2. Espera un tiempo en Centroide Origen (opcional)
3. Caminar a una parada
4. En parada.
   4.1. Espera un tiempo en la parada (vuelve a 4)
   4.2. Caminar a una parada (volver a 3)
   4.3. Tomar ómnibus
5. Bajar de ómnibus
   5.1. Llega a parada (volver a 4)
   5.2. Parada Final (destino pasajero)
6. Caminar a Centroide Destino
7. Fin del ciclo de vida.

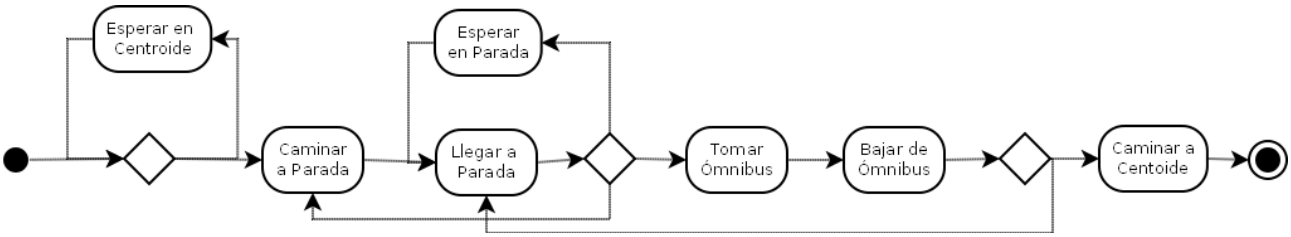


Figura 3 - Ciclo de vida de un pasajero

Cabe destacar que el ciclo de vida definido soporta la realización de transbordos para efectuar el viaje.

3.2.2 - Simulador

En el sistema de transporte urbano de pasajeros, donde existen muchas interacciones internas y un alto uso de recursos importantes (ómnibus por ejemplo), es necesario utilizar técnicas de SED, ya que experimentar con el sistema real es costoso e inapropiado. El simulador desarrollado se basa en esta técnica. Permite representar dinámicamente un sistema de transporte público urbano, es decir, los cambios de su estado según el avance del tiempo. Se modela la interacción de los pasajeros y los ómnibus, dado un diseño del sistema de transporte público y un conjunto de hipótesis sobre el mismo, a saber: el ciclo de vida de los pasajeros, un escenario particular de demanda y una determinada configuración de líneas y frecuencias. Más precisamente, permite modelar diferentes comportamientos de los pasajeros en cuanto a la elección de líneas, mediante la posibilidad de implementar diversas estrategias de viaje.

El modelo permite incluir variabilidad en la evolución del sistema, ya que es un modelo estocástico. Por lo tanto, la generación de demanda, la elección de estrategias, las velocidades de los ómnibus y pasajeros, están modeladas con distribuciones de probabilidad con parámetros que pueden ser definidos por el usuario. El efecto de la variabilidad en la evolución del sistema, es medido mediante histogramas para estudiar los tiempos de viaje de los pasajeros.

El mecanismo de avance del tiempo se estructura en dos fases, los procedimientos que describen los eventos fijos incluyen todos lo eventos condicionados que van a suceder como resultado de los propios eventos agendados o fijos. Por lo tanto, todos los eventos definidos son de tipo B (B events, bound), su ocurrencia es predecible y pueden ser agendados.

Entidades

Se definieron dos entidades, una para modelar los pasajeros y otra para modelar los ómnibus del sistema.

Colas del simulador

Para cada parada existen dos colas, una para modelar los ómnibus que están detenidos (se encola un ómnibus cuando hay pasajeros para subir o bajar), y otra para modelar el comportamiento de los pasajeros que están esperando en la parada por el arribo de algún ómnibus de alguna línea. Cada ómnibus creado en el sistema tiene una cola de pasajeros, se encolan en ella los pasajeros que actualmente están viajando en el ómnibus.

Distribuciones

Para modelar la aleatoriedad de arribo de pasajeros, la asignación de estrategias, la velocidad de los ómnibus y la velocidad de caminata de pasajeros, se utilizan distribuciones de probabilidad.

Se supondrá que los arribos de individuos al sistema son independientes entre sí, aleatorios y pueden ocurrir en cualquier momento de la simulación. Por lo tanto, dicho comportamiento es representado por el simulador mediante una distribución Exponencial Negativa. De esta forma se describe el tiempo que transcurre entre ocurrencias aleatorias, y particularmente en este caso, que son arribos de pasajeros al centroide origen. El tiempo de arribos de pasajeros entonces, se modelará con una distribución exponencial negativa con media igual a la demanda entre centroide origen y destino proporcionada en los datos de entrada del simulador, en particular la matriz origen-destino.

Cuando un pasajero es creado en el sistema, este debe tener una estrategia a realizar. Cada estrategia tiene un porcentaje de ocurrencia en el sistema. Para poder asignar una estrategia se utiliza una distribución Discreta General. Una variable aleatoria discreta representa las estrategias con sus respectivas probabilidades. De esta forma se genera un muestreo que mantenga la probabilidad configurada.

La velocidad de los ómnibus no es la misma para todos ellos, ni en cada tramo que realizan en su recorrido. Si bien existe una velocidad media a la que circulan, en la realidad existen distintos factores que pueden hacer que la velocidad varíe con respecto a dicho valor fijo. Aquellas actividades donde la variación de su duración es aleatoria alrededor de un valor medio, son usualmente modeladas por la distribución Normal. Esto implica que los tiempos insumidos por los ómnibus en un tramo de un recorrido, se van a modelar mediante muestras de una distribución Normal (cuya media es igual a la velocidad media de los ómnibus y desviación estándar en 1/5 de la media) y las distancias de los tramos, como se muestra en la ecuación 1. Si la muestra de la normal es cero, entonces tiempoTramo es cero. En la Figura 4 se visualizan los tramos sobre los cuales se realiza este cálculo.

$$tiempoTramo = Distancia(tramo) / distNormal(media,devstd) \quad (1)$$

Para el tiempo insumido en las caminatas de los pasajeros se utiliza la misma lógica: la distancia a recorrer en la caminata sobre la velocidad de caminata dada por una muestra de distribución Normal (cuya media es igual a la velocidad media de caminata y desviación estándar 1/5 de la media), como se muestra en la ecuación 2. En la Figura 4 se visualizan las caminatas sobre las cuales se realiza este cálculo.

$$tiempoCaminata = Distancia(caminata) / distNormal(media,devstd) \quad (2)$$

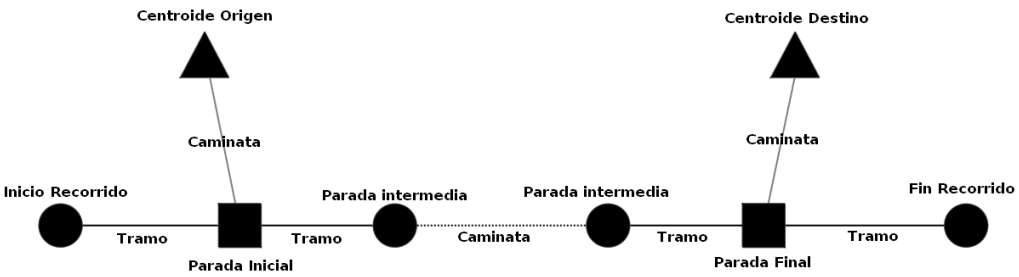


Figura 4 - Ejemplo del viaje de un pasajero sobre un recorrido

Eventos

A continuación se describen los eventos utilizados para modelar el aspecto dinámico del sistema. En primera instancia se muestran los que modelan el comportamiento de los ómnibus y luego los que modelan el comportamiento de los pasajeros. Además, en cada figura se observan cuales son los posibles flujos que pueden tomar las entidades en cuestión (pasajeros / ómnibus). La Figura 5 corresponde al flujo de eventos de los ómnibus, mientras que la Figura 6 al de los pasajeros.

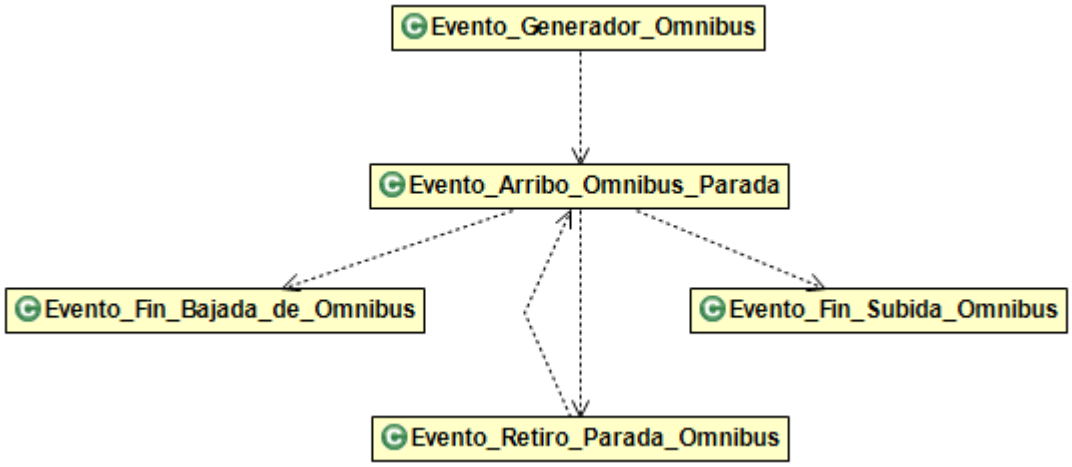


Figura 5 - Flujo de eventos de los ómnibus

Evento_Generador_Omnibus: Evento encargado de generar un ómnibus de una línea. Además agenda en tiempo cero el arribo del ómnibus a su parada inicial (*Evento_Arribo_Omnibus_Parada*). Se agenda la ejecución de este mismo evento según el tiempo especificado por la tabla de horarios de la línea.

Para los siguientes tres eventos se asume que todos los pasajeros toman el mismo tiempo en subir y bajar del ómnibus, y lo hacen en el mismo momento. Además se asume que ese tiempo (al que se llamará tiempo_retiro_de_parada) es el que permanece el ómnibus en la parada en caso de que haya pasajeros para subir o bajar.

Evento_Arribo_Omnibus_Parada: Evento encargado de modelar el comportamiento cuando un ómnibus arriba a una parada de su recorrido. Verifica si hay pasajeros para descender en dicha parada. Si es así, agenda en tiempo_retiro_de_parada el evento que marca el descenso de un pasajero del ómnibus (*Evento_Fin_Bajada_de_Omnibus*). Además, mientras la capacidad del ómnibus lo permita, verifica si hay pasajeros en la parada que deseen subirse al ómnibus. Para cada uno de ellos, agenda en tiempo_retiro_de_parada el evento que marca el ascenso de un pasajero al ómnibus (*Evento_Fin_Subida_de_Omnibus*). Se agenda el retiro del ómnibus (*Evento_Retiro_Parada_Omnibus*) de la parada en tiempo_retiro_de_parada.

Evento_Fin_Subida_de_Omnibus(omnibus, pasajero): Evento encargado de efectivizar el ascenso de un pasajero a un ómnibus, quita al pasajero de la cola de pasajeros de la parada y lo agrega en la cola de pasajeros del ómnibus correspondiente a sus necesidades.

Evento_Fin_Bajada_de_Omnibus(omnibus, pasajero): Evento encargado de efectivizar el descenso de un pasajero de un ómnibus, quita al pasajero de la cola de pasajeros del ómnibus en que viajaba y agenda en tiempo cero el arribo del pasajero a su parada final para dicho recorrido (*Evento_Arribo_Pasajero_Parada*), es decir, se asume que no transcurre tiempo.

Evento_Retiro_Parada_Omnibus: Evento encargado de modelar el comportamiento para poner a transitar un ómnibus entre paradas de su recorrido. Maneja las colas de ómnibus de una parada y la cola de ómnibus en tránsito. Agenda, en tiempo de viaje entre dos paradas (calculado a partir de la distancia entre ellas y de la velocidad dada por la distribución de probabilidades) el evento de arribo de un ómnibus a una parada (*Evento_Arribo_Omnibus_Parada*).

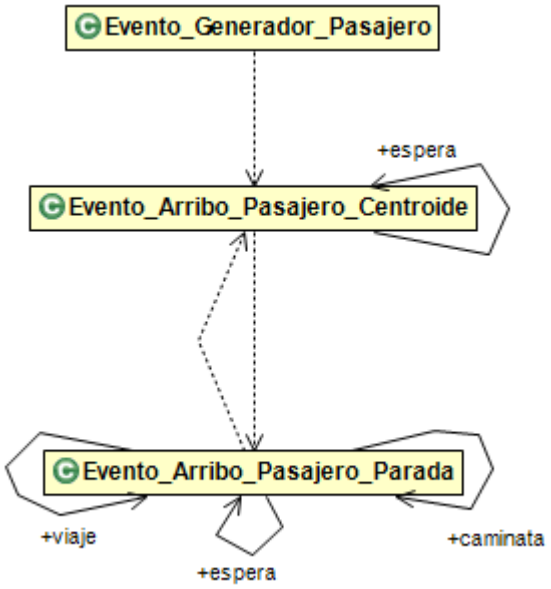


Figura 6 - Flujo de eventos de los pasajeros

Evento_Generador_Pasajero: Evento encargado de generar un pasajero para un par de centroides origen-destino. Además agenda en tiempo cero el arribo del pasajero a su centroide origen (*Evento_Arribo_Pasajero_Centroide*). Se agenda la ejecución de este mismo evento según el tiempo dado por la distribución para el par origen-destino, para generar una nueva entidad pasajero para dicho par.
Evento_Arribo_Pasajero_Centroide: Evento encargado de modelar el comportamiento cuando un pasajero arriba a su centroide origen. El comportamiento de este evento depende de la elección del pasajero. Las opciones son:

- si el pasajero desea esperar un tiempo en el centroide, se agenda en dicho tiempo nuevamente el arribo del pasajero al centroide (*Evento_Arribo_Pasajero_Centroide*).
- si el pasajero desea caminar hacia una parada, se agenda en tiempo de caminata a la parada final el arribo del pasajero a la parada (*Evento_Arribo_Pasajero_Parada*).

Evento_Arribo_Pasajero_Parada: Evento encargado de modelar el comportamiento cuando un pasajero arriba a una parada. El comportamiento de este evento depende de la elección del pasajero. Las opciones son:

- si el pasajero desea esperar un tiempo en la parada, se agrega el pasajero en la cola de espera de pasajeros de la parada y se agenda en tiempo de espera del pasajero nuevamente el arribo a la parada (*Evento_Arribo_Pasajero_Parada*).
- si el pasajero desea caminar hacia otra parada, quita al pasajero de la cola de pasajeros de la parada. Dado el tiempo de caminata desde la parada que se encuentra hasta la parada final, se agenda el arribo del pasajero a la parada final de la caminata (*Evento_Arribo_Pasajero_Parada*).
- si el pasajero desea caminar hacia un centroide, quita al pasajero de la cola de pasajeros de la parada. Dado el tiempo de caminata desde la parada que se encuentra hasta el centroide destino, se agenda el arribo del pasajero al centroide destino de la caminata (*Evento_Arribo_Pasajero_Centroide*).
- si el pasajero desea tomar un ómnibus, agrega el pasajero en la cola de espera de pasajeros de la parada. Registra para el pasajero en cuestión cuáles son las líneas que le sirven para completar su viaje. En este caso no se agenda otro evento, cuando un ómnibus arriba a una parada (*Evento_Arribo_Omnibus_Parada*) él es el encargado de verificar si hay pasajeros que deseen subirse al ómnibus (estos estarán en la cola de espera de la parada y tienen registrado como posible línea a tomar la línea correspondiente al ómnibus que arriba a la parada).

Histogramas

Se definió un histograma con el objetivo de poder medir el tiempo promedio que demoran los pasajeros desde su creación en su centroide de origen hasta que llegan al centroide destino.

Una vez que los pasajeros finalizan su viaje en el centroide destino, registran el tiempo total en el histograma. A partir de este histograma es posible calcular el tiempo de viaje promedio de un pasajero en la simulación. El valor medio de este histograma se utiliza como el indicador principal de comparación entre los distintos casos de estudio. Además, se definió otro histograma con las mismas características pero que acumula los datos para cuando la simulación está compuesta por más de una replicación.

3.2.3 - Estrategias (modelo del comportamiento de los pasajeros)

Un pasajero puede tener diferentes formas de planificar un viaje de un origen a un destino. Las estrategias son un punto importante en el desarrollo de esta herramienta. A partir de ellas, se pueden realizar varios estudios sobre las diferentes decisiones de los pasajeros.

El ciclo de vida de un pasajero es la base de todo el sistema. A partir de éste, se desprenden las posibles acciones a realizar por cualquier pasajero en los diferentes momentos de su viaje. Las acciones posibles son: esperar determinado tiempo en el centroide origen, caminar a una parada, esperar un tiempo determinado en una parada, tomar un determinado ómnibus y bajar en una parada del recorrido, y caminar al centroide destino.

Con una correcta configuración de las acciones mencionadas, se puede modelar cualquier viaje de un pasajero para ir desde un origen a un destino. Es la función de la estrategia, planificar un conjunto de viajes posibles, tomar las decisiones e informar en cada momento al simulador cual es la siguiente acción a ejecutar según esa decisión.

El cometido de las estrategias es que cumplan con determinada característica, como por ejemplo minimizar la caminata, minimizar el tiempo de viaje sin información de horarios, minimizar el tiempo de viaje con información estática de horarios, minimizar tiempo de viaje contando con información del sistema en tiempo real sobre horarios, minimizar recorrido de viaje según la distancia recorrida, minimizar número de transbordos, tomar el primero que llegue a la parada que pueda servir, llegar a destino cercano a determinada hora, tomar determinada línea, etc.

Una estrategia puede planificar uno o varios viajes al momento de crearse un pasajero y mantener esta planificación durante todo el ciclo de vida del pasajero, o puede re-planificar en cada momento del viaje según el estado del sistema.

Al crearse un pasajero se le asigna una estrategia a realizar. Esta planea el o los viajes según la información con la que cuenta. Cada vez que el simulador ejecuta algún evento de un pasajero, le consulta a este cuál es la próxima acción a realizar, es en ese momento que la estrategia tiene la opción de re-planificar su viaje o no.

Es posible agregar las estrategias que sean necesarias para poder realizar diferentes análisis sobre un caso de estudio. También es posible indicar qué porcentaje de cada estrategia es asignado a los pasajeros.

En este proyecto se desarrollaron dos estrategias particulares, las cuales se describen a continuación. El objetivo de las mismas es mostrar que el simulador desarrollado soporta distintas implementaciones de estrategias.

Minimizar tiempo de viaje maximizando tiempo de espera en centroide

El objetivo de la estrategia es que el pasajero pueda esperar en el centroide de origen sabiendo que va a llegar al centroide destino en el menor tiempo posible teniendo en cuenta las distintas combinaciones de líneas del sistema. Esto se basa en la hipótesis de que el tiempo en el origen es valioso, es decir, el pasajero puede hacer algo productivo en la casa, mientras que el tiempo en la parada es perdido.

Sabiendo cuales son los horarios de las líneas de ómnibus, el pasajero puede calcular cual es el ómnibus que tiene que tomarse para realizar el viaje en el menor tiempo posible, y también el mayor tiempo que puede quedarse en su casa antes ir a la parada.

La estrategia minimizará el tiempo de espera ocioso en parada permitiéndole al pasajero esperar en el centroide. Retorna para un centroide origen (CO) y un centroide destino (CD) dependiendo del instante de tiempo:

- tiempo de espera en centroide origen
- parada inicial (PI) a la cual se debe caminar
- conjunto de ómnibus que puede tomar el pasajero y cuál es la parada destino para cada uno de ellos

El cálculo del tiempo para aceptación del viaje se puede especificar con la ecuación 3:

$$min\{max\{espera\ en\ CO\} + tiempo\ caminata\ a\ PI + tiempo\ de\ traslado + tiempo\ caminata\ a\ CD\} \quad (3)$$

Minimizar Caminatas

El objetivo de la estrategia es que el pasajero camine lo menos posible teniendo en cuenta las caminatas desde el centroide origen (CO) a la parada inicial (PI) y desde la parada final (PF) al centroide destino (CD). La estrategia retorna para un centroide origen y un centroide destino:

- parada inicial a la cual se debe caminar
- conjunto de ómnibus que puede tomar el pasajero y cuál es la parada destino para cada uno de ellos

El cálculo del tiempo para aceptación del viaje se puede especificar con la ecuación 4:

$$min\{distancia\ de\ CO\ a\ PI + distancia\ de\ PF\ a\ CD\} \quad (4)$$

3.2.4 - Visualizador

El visualizador es una representación gráfica del simulador mediante registro de información. Esto es, que el simulador le informa al visualizador los cambios generados durante los eventos ejecutados. Este último a partir de esa información, realiza la representación gráfica correspondiente al estado del sistema, haciendo que esté sincronizado con el simulador y por lo tanto mostrando en forma simultánea la simulación. El visualizador consta de componentes estáticos obligatorios (paradas, centroides, zonas, líneas de ómnibus, caminatas), componentes estáticos opcionales (mapa de la ciudad, calles, etc) y componentes dinámicos como lo son los pasajeros, los ómnibus y las estrategias.

Entre las principales funcionalidades del visualizador se encuentran:

- Carga inicial de datos.
- Visualización de la simulación.
- Agregar capas extra al mapa.
- Seguimiento de un pasajero

A continuación se describen dichas funcionalidades.

Carga inicial de datos

Esta funcionalidad permite la carga de datos para la simulación y visualización de los casos de estudio del sistema, los cuales fueron generados en el proceso de transformación de datos de entrada.

Los datos necesarios son: paradas de ómnibus, recorridos de ómnibus, zonas de demanda, horarios de ómnibus, frecuencia de salidas de origen a destino de pasajeros y datos de configuración como la velocidad media de los ómnibus y las personas, la capacidad del ómnibus, los porcentajes de diferentes estrategias a utilizar y el tiempo de ejecución de la simulación.

Alternativamente es posible ejecutar la simulación sin la visualización y realizar replicaciones para hacer análisis sobre los casos de estudio. Además, se permite deshabilitar las distribuciones utilizadas en el simulador para realizar validaciones y estudios específicos. En ese caso se utilizarán datos determinísticos.

Visualización de la simulación

La visualización tiene que ser amigable y entendible a simple vista. Cuenta con elementos fijos, como lo son las paradas, las zonas con sus respectivos centroides, las caminatas de los centroides a las paradas, los recorridos de cada línea, y los elementos dinámicos que son los ómnibus, los pasajeros, el avance y el tiempo de la simulación.

Los elementos fijos son cargados desde la interfaz de carga de datos. Es posible modificar los estilos, ocultar y ordenar la forma en que se muestran estos elementos del mapa.

Los ómnibus y los pasajeros son creados y manipulados desde el simulador, indicando al visualizador cual es el estado en que se encuentra cada instancia. Es posible ocultar estos elementos del mapa. También desde el simulador se informa cual es el estado de la simulación.

Es posible pausar la simulación y retomarla en el momento que se necesite o finalizar antes del tiempo configurado.

Al principio de la simulación o cada vez que se pausa la misma, es posible modificar la velocidad con la que se ejecuta. La velocidad que se puede configurar va desde 0 hasta 41 veces la velocidad de simulación. La velocidad máxima de visualización fue elegida para que no resulte incómoda a la vista, ya que si es demasiado rápida no se podrá entender cómo transcurre la simulación -esto está sujeto al hardware utilizado-.

También es posible hacer replicaciones de una simulación, para realizar análisis estadísticos sobre el caso de estudio en cuestión. Cuando se ejecuta esta funcionalidad, no se realiza la visualización de la simulación y la velocidad de la misma es la máxima posible. Además, en vez de mostrar el tiempo de la simulación, se muestra el número de la replicación en que se encuentra.

Agregar capas extra al mapa

Además de las capas necesarias para el funcionamiento del sistema, es posible agregar capas extra al mapa para una mejor visualización (componentes estáticos opcionales). Queda a criterio del usuario final si desea o no agregar dichas capas. Sobre estas capas no hay ningún control del sistema ya que su función es como complemento de la visualización.

Seguimiento de un pasajero

Es necesario poder realizar el seguimiento de un pasajero particular. A partir de un centroide origen y un centroide destino, se elige el n-ésimo pasajero a hacerle seguimiento de su recorrido y decisiones, mostrando la estrategia que tiene para realizar. La selección puede ser en cualquier momento de la simulación, siendo el n-ésimo pasajero del centroide origen seleccionado, a partir del momento de la selección.

La estrategia se muestra al momento que se crea el pasajero a seguir y se elimina cuando termina el ciclo de vida del mismo. Se muestran los recorridos posibles, discriminando entre caminatas, recorridos de ómnibus y transbordos realizados.

Capítulo 4 - Datos de entrada

La aplicación desarrollada en el presente proyecto requiere de datos de entrada para su ejecución. En este capítulo se presenta cuáles y cómo deben ser los datos para un funcionamiento correcto. Previamente se hace referencia a un juego de datos base -el caso de Rivera-, y se lo utiliza para establecer los requerimientos de la aplicación. Además se expone detalladamente cómo se diseña e implementa el proceso de transformación de datos -que forma parte de la herramienta-, así como sus requerimientos específicos y sus objetivos. La última sección, está dedicada a los datos del caso de Montevideo-Uruguay, ya que uno de los objetivos del proyecto es que la herramienta se comporte con casuísticas similares en cuanto a volumen de datos.

4.1 - Datos base - Caso de Rivera

El Departamento de Investigación Operativa cuenta con datos de la ciudad de Rivera-Uruguay, los cuales han sido insumo en proyectos anteriores [1][2][6][7]. Dado que se cuenta con estos y que se conoce la consistencia de la información que contienen, son considerados como datos base para este proyecto. Dicho juego de datos comprende archivos con propiedades espaciales como: un shapefile de paradas, un shapefile por cada recorrido de líneas de ómnibus y un shapefile de zonas; y archivos planos con información de la frecuencia de pasajeros origen-destino (matriz O-D). Sus dimensiones permiten la experimentación, y -como ya ha sido objeto de otros proyectos- permite tener una referencia de los valores esperados de trabajar con él. Este juego de datos también permite establecer una noción de límites en cuanto a exigencias de entrada para otros juegos de datos.

4.2 - Transformación de datos

En la sección *3.1* se establece la existencia de una parte de la solución del proyecto como lo es el proceso de transformación de datos, así como sus características, necesidades y objetivos. En esta sección, se propone estudiar dicho proceso del punto de vista de: sus componentes, requerimientos, implementaciones, plataforma de desarrollo y funcionamiento.

Uno de los objetivos de este proyecto es una herramienta solución, que pueda funcionar para cualquier juego de datos que cumpla con ciertas condiciones que se detallan en esta sección. Como base del estudio se utiliza los datos de Rivera. Las características de estos datos ayudan a establecer exigencias mínimas que deben de cumplir cualquier juego de datos de entrada. Sin embargo, no son exigencias excluyentes para otro juego de datos.

Por otra parte, otro de los objetivos inherente a los datos de entrada es minimizar la exigencia al usuario sobre la modificación de su formato, aumentando la compatibilidad del sistema. Ejemplo de ello son los identificadores utilizados por la aplicación, los cuales se generan automáticamente con este proceso.

Para la transformación se tuvo en cuenta las necesidades del sistema para su correcto funcionamiento y los requerimientos del proyecto.

Las necesidades del sistema son:

- contar con un único shapefile de tipo línea de todas las líneas de ómnibus, conteniendo:
  - un identificador por línea de ómnibus, éste es compartido por los distintos tipos de recorridos de una misma línea de ómnibus.
  - un identificador único de recorrido
  - un atributo que identifique de qué tipo es: ida, vuelta o circular
  - un conjunto de paradas válidas, donde deben guardar una relación de orden de forma que permita definir en qué orden se recorren
- contar con un único shapefile de tipo polígono, que contenga zonas que delimitan áreas de creación

y de arribo de pasajeros. Además, dichas zonas definen el área donde un pasajero puede “caminar”.
    - cada zona debe de contar con un identificador.
- contar con un único shapefile de puntos que representan las paradas del sistema
    - cada parada debe tener un identificador.

Los requerimientos del proyecto son:

- contar con una solución que permita agregar paradas específicas al recorrido de una línea en particular.
- Soportar como datos de entrada, aquellos con características similares a los existentes de Rivera. La característica de éstos, se asumirán como requerimiento del sistema, como ser la nomenclatura de los archivos, el formato de éstos, etc.

Dado lo expuesto en los puntos anteriores y las motivaciones establecidas en la sección *3.1*, la solución encontrada converge en la definición de un proceso compuesto por distintos componentes de ejecución manual. Cada uno tiene un propósito que ataca alguna o varias de las necesidades anteriores, pero su ejecución guarda independencia de las restantes. Esto permite ejecutar sólo aquellos componentes que hagan falta -o su totalidad en caso de ser necesario- ganando flexibilidad en las exigencias al usuario.

4.2.1 - Diseño e implementación

Gran cantidad de los datos de entrada son georeferenciados, es así que se decidió explotar el potencial del paradigma SIG de forma que cubra las necesidades del sistema. Sin embargo, es prioridad que el proceso pueda ser automatizado -en la medida de lo posible-, y que el usuario no deba ser un experto SIG para poder utilizarlo.

Se estudiaron soluciones que brindan las herramientas SIG más destacadas en este sentido, y que redundan en la ejecución y desarrollo de scripts que creen nuevos shapefiles pero que mantengan los datos originales y satisfagan todos los requerimientos. Dadas las valoraciones detalladas en *Anexo 1 - Decisiones sobre herramienta de transformación de datos,* se utilizó la herramienta *gvSIG* para tal fin y en particular su módulo de scripting.

Para el procesamiento se definieron cinco componentes, cada uno compuesto por un script: *unificarLineas.py, agregarParadasLineaFile.py, setParadaLineaSeleccion.py, crearIdParada.py y crearIdZona.py*. Estos hacen uso del módulo de scripting de *gvSIG* y su entorno de ejecución. Se implementaron en *Jython*, una adaptación que permite ejecutar programación de *Python* con un programa que trabaja sobre el lenguaje de Java.

La solución general de cada script es crear nuevos shapefiles, que mantengan la geometría de los originales, agregando a cada esquema los atributos necesarios para cumplir los requerimientos de la herramienta y otros que se consideren de interés.

A continuación se expondrá cada script desarrollado. En los casos que se creen conveniente se incorpora una figura esquemática de su funcionamiento para facilitar la comprensión.

unificarLineas.py

Este componente es el encargado de unificar en un único shapefile los recorridos de las líneas de ómnibus en caso de que se tenga un shape por recorrido, como lo es el caso de Rivera.

Los archivos shapefile esperados deben cumplir requerimientos que se exponen a continuación:

- Cada shape es una representación de un único recorrido de línea de ómnibus.
- Los archivos de entrada deben seguir la siguiente nomenclatura:
    - deben poseer el prefijo “Linea” (o cualquier combinación de 5 caracteres), seguido por el nombre de la línea, concatenado con la palabra “Desde” y debe poseer un sufijo que podría ser el origen de donde sale la línea.

El script que se ejecuta en este componente, toma todos los archivos shape (en tiempo de ejecución el

script sólo toma en cuenta los archivos con extensión “shp”) que estén contenidos en una determinada carpeta -asumiendo que cumplen los requerimientos expuestos-, para generar un shape nuevo que consolide todos los archivos en uno solo. Además, al esquema del archivo de salida se le agregan atributos necesarios como son:

“*TIPO”* - Es el tipo de recorrido de la línea y es de tipo “String”

“*ID”* - Es un autoincremental que identifica al recorrido en el Sistema, es del tipo “Integer”

“*ID_LINEA”* - Identifica a la línea de ómnibus y es del tipo “String” aunque tome valores numéricos enteros

“*PARADAS”* - Almacena los identificadores de las paradas de un recorrido. Es de tipo “String”. Este script no agrega datos en dicho atributo, este campo es completado por el componente *agregarParadasLineaFile.py*.

Se toma a cada archivo cargado como una única entidad Línea sin importar si es del tipo Línea o Polilínea y se agrega a un shape de salida luego de su procesamiento. El procesamiento consta de -para cada shapefile- fundir todas las eventuales geometrías en una sola entidad, a la cual se le asigna un identificador autoincremental (“*ID”*). Para ésto, se une geometría a geometría, recorriendo cada una de estas desde el punto de vértice 0 hasta el último de cada una. En caso de ser una Polilínea, es decir que haya más de una entidad, se asume que están en orden. Esto es, la Línea que se toma primero y cual le sigue son consistentes con el recorrido de la línea de ómnibus. Se entiende que esta última asunción es muy básica para incluirla en los requerimientos pero bien vale un comentario. Este es el caso de los archivos de Rivera, donde además se presenta la complejidad que no mantienen una continuidad exacta entre Polilíneas, problema que también se ve resuelto por el script. Haciendo uso del nombre de cada archivo, se identifica de qué tipo es cada línea. Si en la carpeta seleccionada hay más de un archivo de linea con el mismo origen, se le asocia arbitrariamente el atributo “*TIPO”* uno como ”IDA” y al otro “VUELTA” además de un identificador de línea compartido (“*ID_LINEA”*). De lo contrario, si hubiera un solo archivo con ese nombre de línea, se le asigna el tipo “CIRCULAR” junto con el identificador de la línea correspondiente. El resultado del procedimiento anterior será una entidad del tipo Línea que comprende todo el recorrido de una línea de ómnibus, y por tanto es contínua y tiene un sentido determinado. Dicha entidad -ya procesada- se agrega al shapefile de salida.

El archivo de salida resultado de la ejecución contendrá cada nueva entidad procesada. El script no reemplaza archivos, por lo que si ya existe uno con el mismo nombre del de salida se pedirá que ingrese otro nombre válido.

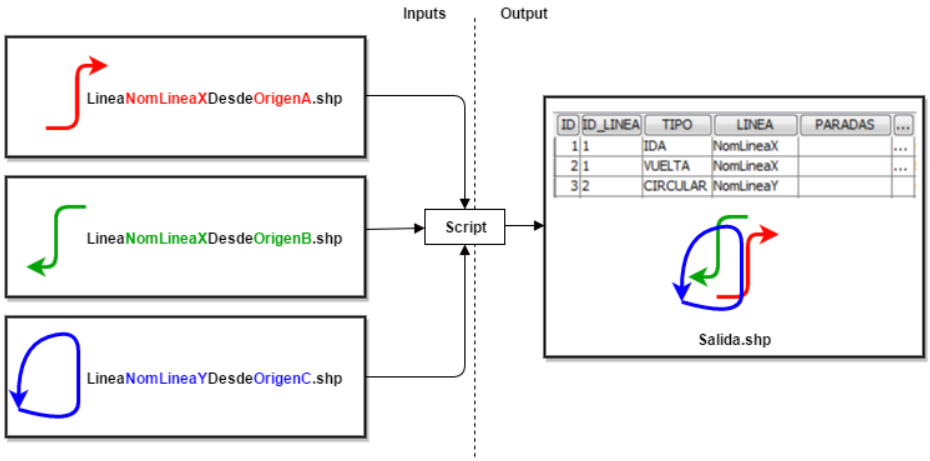


Figura 7 - Esquema del funcionamiento de unificarLineas.py

En la Figura 7 se muestra un esquema del funcionamiento del componente a partir de los datos de entrada. A la izquierda del mismo los inputs: los shapes de recorridos; a la derecha el output: la unificación de la entrada.

agregarParadasLineaFile.py

Este componente complementa el anterior. Espera como entrada un shapefile y un archivo plano. El shapefile es obtenido al ejecutar el script *unificarLineas.py*, que contiene un atributo "PARADAS" en su esquema. En el archivo plano de entrada se asume una fila por entidad Línea (correspondiente a un recorrido). Cada fila empieza con el id de un recorrido, seguido por un tabulador y una secuencia de identificadores de paradas separadas por ";".

El script de la componente -*AgregarParadasLineaFile.py*- toma el archivo plano correctamente tabulado y busca la primer secuencia de caracteres. Esta deberá ser el identificador de una entidad en el shapefile de entrada ("*ID"* de recorrido). Luego busca la siguiente serie de caracteres en el archivo plano y se la asigna al atributo "PARADAS" de dicha entidad. Cabe acotar que la noción de orden de las paradas está dada por el recorrido de un ómnibus.

Este proceso se repite fila por fila del archivo de texto. Si una entidad contara con datos en dicho atributo, este se reescribirá con el string asociado al id del recorrido.

La salida del script es el shapefile de entrada con el atributo "PARADAS" modificado para cada entidad. La Figura 8 muestra un esquema del funcionamiento del componente a partir de los datos de entrada. A la izquierda los inputs: el shape y el texto plano de paradas; a la derecha el output: el mismo shape de entrada pero actualizando sus paradas.

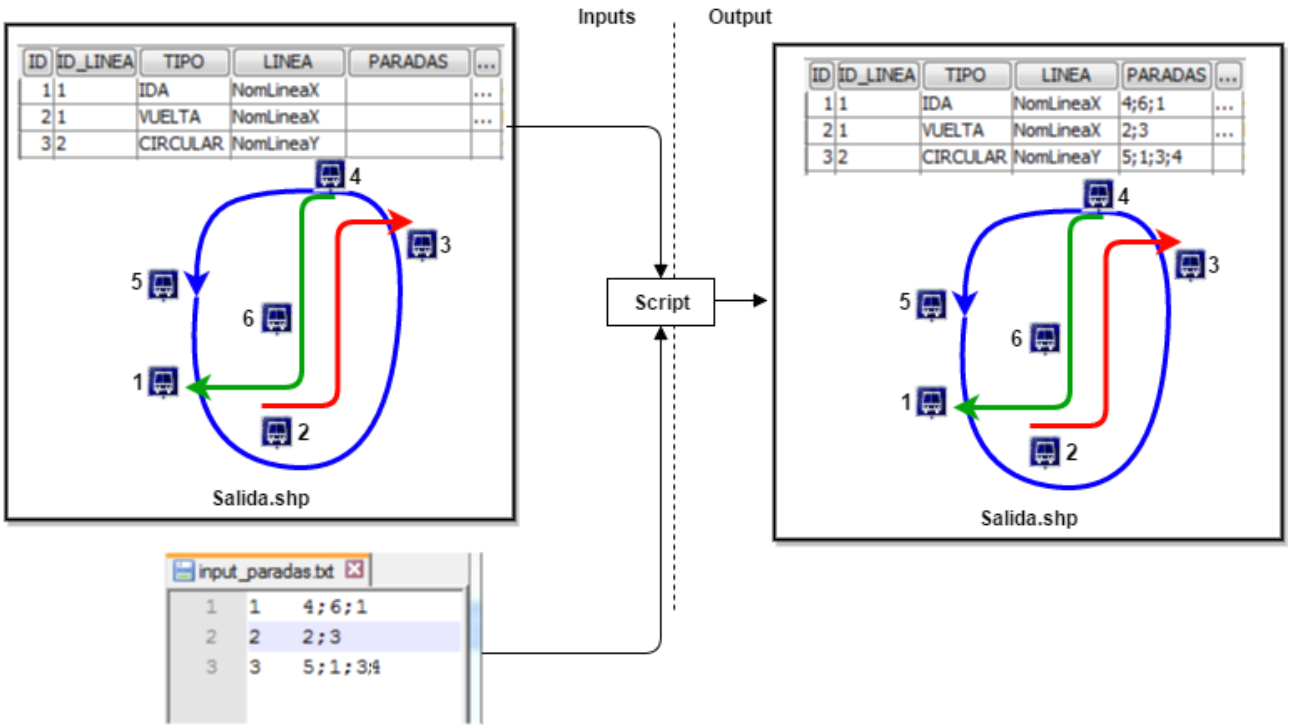


Figura 8 - Esquema del funcionamiento de agregarParadasLineaFile.py

setParadaLineaSeleccion.py

El script que contiene esta componente, es una versión manual del anterior y con una orientación más gráfica. Espera un shape de líneas y otro de paradas, y devuelve el mismo shape de líneas modificado. La implementación identifica una entidad línea ingresada por el usuario y previamente, seleccionando una parada desde el área de datos gráficos de *gvSIG*, y concatena al final el identificador de la parada elegida (en su modo gráfico) al atributo “PARADAS” de la entidad identificada.

Es posible seleccionar varias paradas en una misma ejecución, aunque no siempre se guarda el orden de la selección, por lo que no se recomienda esta práctica.

crearIdParada.py

El script toma una capa de paradas y modifica su esquema agregando un atributo: “ID_PARADA”. Luego actualiza ese atributo para cada entidad con un identificador autoincremental.

crearIdZona.py

El script toma una capa de zonas y modifica su esquema agregando un atributo: “ID_ZONA”. Luego actualiza ese atributo para cada entidad con un identificador autoincremental.

4.2.2 - Archivos de entrada - Configuración y composición

Luego del proceso descrito en la subsección *4.2.1*, en esta subsección se exponen los datos específicos que requiere el simulador. Se lista cada archivo y se comenta su objetivo y los detalles de su composición.

Shapefile de paradas.

Con este archivo se cargan las paradas del sistema, por lo que debe ser del tipo geográfico punto.
El único campo necesario dentro de su esquema es el atributo “ID”, que opera como identificador y es del tipo numérico. Es el archivo generado luego del proceso “crearIdParada.py”.

Shapefile de zonas.

Es necesario contar con un archivo geoespacial que corresponda a zonas de concentración de población, por lo que debe ser del tipo polígono. A partir de estas zonas se crean centroides de donde se lleva a cabo la creación y destrucción de pasajeros.

El campo necesario en el shapefile de zonas es el ID que identifica en forma única a cada zona, que es el mismo para el centroide de la zona. Los ID de las zonas tienen que ser consecutivos y empezar en cero, esto es debido a que tienen correspondencia directa con la matriz OD. Es el archivo generado luego del proceso “crearIdZona.py”.

Shapefile de líneas de ómnibus.

Este archivo geoespacial representa las líneas de ómnibus con sus recorridos. El archivo de líneas de ómnibus, debe contener los siguientes campos: ID (identificador numérico único de la capa de línea), ID_LINEA (nombre de la línea de ómnibus), TIPO (corresponde a si el recorrido de la línea de ómnibus es ida, vuelta o circular) y PARADAS (identificadores ordenados de las paradas del recorrido de la línea de ómnibus, cada parada debe estar separada por “;”). Este último campo no es obligatorio si en la ruta del shapefile, se encuentra un archivo de nombre “*Recorridos.csv”*.

El archivo “*Recorridos.csv”* debe tener siguiente formato:

Cabezal : ID, Paradas

Cuerpo: “*ID”, “NroParadas”*

Donde “*ID”* corresponde al identificador numérico único de la capa de línea y “*NroParadas*” a los identificadores ordenados de las paradas del recorrido de la línea de ómnibus, cada parada debe estar separada por “;”.

Un ejemplo de "*Recorridos.csv"* es:

ID,Paradas
1,3010;3011;4839;2913;4840;4893;2915
2,2118;2119;4457;555;2120;3369;3370
3,1753;1754;1755;1756;1757;1955;1956
4,4757;3242;2207;2208;2209;2210;4993
5,2954;2955;2956;2957;2816;2817;2818
6,4757;3242;2207;2208;2209;2210;2211
7,4770;4771;4772;5108;5109;4549;4023
8,1990;1991;2350;2351;2352;2353;2354

El shapefile de líneas de ómnibus es el archivo generado luego del proceso "***unificarLineas.py"*** y "***agregarParadasLineaFile.py"*** o "***setParadaLineaSeleccion.py"***.

Archivo de configuración inicial

Este es un archivo plano y es el mayormente responsable de la parametrización de sistema. No es obligatorio, ya que de no encontrarse se cargan los valores configurados por defecto dentro del sistema.

El archivo debe tener los siguientes parámetros:

- Velocidad media y la desviación estándar del pasajero al caminar (en km/h). En caso de que la desviación estándar sea 0, todos los pasajeros tienen la misma velocidad de caminata (velocidad media).
- Velocidad media y la desviación estándar del ómnibus (en km/h). En caso de que la desviación estándar sea 0, todos los ómnibus tienen la misma velocidad (velocidad media).
- Capacidad del ómnibus. Todos los ómnibus tienen la misma capacidad de pasajeros.
- Tiempo de demora del ómnibus en la parada si tiene que frenar (en minutos).
- Duración de la simulación (en minutos).
- Sistemas de referencia de coordenadas para las capas a utilizar.
- Cantidad de replicaciones del experimento. Este campo es utilizado para generar varias replicaciones de un experimento.
- Porcentaje de las estrategias a utilizar. Se especifica para cada estrategia el porcentaje de asignación a usuarios, es decir, la probabilidad de que a un usuario se le asigne cierta estrategia.
- Utilizar la distribución para la creación de pasajeros. Este valor booleano corresponde a si el simulador debe o no utilizar la distribución Exponencial Negativa para el arribo de pasajeros.
- Utilizar la distribución para la elección de estrategias a asignar a cada pasajero. Si el valor es true, se utiliza una distribución con diferente semilla en caso de que se ejecute varias replicaciones del caso de estudio.

El formato del archivo se ve en el siguiente ejemplo, aunque no tiene que cumplir el orden que se muestra.

VelocidadPersona 3.0 0.6
VelocidadOmnibus 10.0 2.0
CapacidadOmnibus 20
TiempoOmnibusParada 1
TiempoDeEjecucion 120
CRS EPSG:32721
estrategias Estrategia_Minimizar_Caminata 50 Estrategia_Minimizar_Tiempo_Viaje 50
Replicaciones 1
DistribucionPasajeros true
DistribucionEstrategias true

Archivo de texto de matriz OD

Cada elemento de la matriz OD representa la tasa (la unidad es 1/minutos) para la creación de pasajeros desde un centroide origen O a un centroide destino D. Es decir, se crea un pasajero cada 1/matriz[O][D] minutos en promedio para ir de O a D. Las filas de la matriz representan el origen y las columnas el destino. Esto es posible ya que como se menciona en los shapefile de zonas, los identificadores de zonas y centroides son consecutivos empezando en cero.

El formato del archivo es un texto plano, donde las filas se separan por salto de línea y las columnas separadas por espacios.

Ejemplo de matriz OD de 4 x 4:

0.000000 0.000000 0.081818 0.230080
0.000000 0.000000 0.000000 0.000000
0.013333 0.000000 0.000000 0.018182
0.000000 0.200000 0.000000 0.000000

Archivo de texto de horarios de ómnibus

El último archivo necesario para cargar el sistema es el que contiene los horarios en que los ómnibus parten de la parada inicial. Hay dos formas para cargar el horario: frecuencia de salida o tabla de horarios.

- Frecuencia de salida: Indica cada cuantos minutos sale un ómnibus de determinada línea.
- Tabla de horarios: Indica todos los horarios de salida de los ómnibus en el día.

El formato del archivo es el siguiente:

ID TIPO [frecuencia | horarios]

donde ID corresponde al nombre de la línea (ID_LINEA de la línea de ómnibus), TIPO es el tipo de recorrido (IDA, VUELTA, CIRCULAR), frecuencia es el tiempo en minutos de cada cuanto sale un ómnibus y horarios es cada uno de los instantes que sale cada ómnibus para la línea separada por espacios con el formato HH:mm.

En el siguiente ejemplo se muestran las formas descritas, en la primera fila se ve un ejemplo de frecuencia y en la segunda uno de horarios:

Linea1 IDA 30
Linea2 VUELTA 00:00 00:45 02:30 15:20

4.3 - Montevideo

Como se establece en la sección *1.2 - Objetivos* en este documento, es de interés analizar el comportamiento del sistema con datos de la ciudad de Montevideo. Para lo cual se recolectaron datos que no respetan los requerimientos descritos en este capítulo y que exigieron un preprocesamiento particular descrito en *Anexo 4 - Datos para el caso de estudio de Montevideo*.

Dicho preprocesado, solo dejó pendiente la creación de los identificadores de la capa de paradas, que se resuleve con la ejecución del script crearIdParada.py completando los requerimentos necesarios para la puesta en marcha del sistema.

Si bien se tienen horarios de ciertas líneas de Montevideo, realizar el procesamiento para adaptar estos datos al formato de entrada de la herramienta no fue un requerimiento en el marco de este proyecto. Por lo que, para cada recorrido se utiliza una salida de ómnibus cada treinta minutos.

En el caso de la matrizOD, no se dispone de datos reales para este caso al momento de la realización de este estudio. La generación de esta no cumple con ninguna condición en especial, se agregan valores aleatorios entre cero y uno a determinados pares OD. Con esto es suficiente para las pruebas a realizar.

Para el archivo de configuración inicial, se utiliza el mismo de la ciudad de Rivera.

Capítulo 5 - Diseño e implementación de la aplicación

En este capítulo se muestra el desarrollo de la aplicación que permite realizar la simulación con su respectiva visualización. Se hace una comparación con las aplicaciones anteriores desarrolladas en el grupo de investigación y las decisiones de diseño que fueron tomadas para cumplir con las funcionalidades requeridas. El capítulo está compuesto de seis secciones. La primera es una reseña de la arquitectura utilizada para cumplir con los requerimientos, la que se contrasta con la arquitectura de los proyectos anteriores. La segunda sección consta del diseño e implementación del simulador, explicando cómo funciona este componente como una entidad independiente. Si bien desde la visualización se carga el sistema y se visualiza y manipula la simulación, la carga del sistema se documenta en una sección aparte, ya que hay conceptos que necesitan ser abordados anteriormente para su comprensión. Por lo que en la sección *5.3* se explica la forma en que funciona la visualización, ya suponiendo el sistema cargado. En la sección *5.4* se describe cómo se realiza la carga del sistema. En las últimas dos secciones, se detalla la integración entre el simulador y el visualizador. En la sección *5.5* se describen las funciones básicas de la integración, como lo son: iniciar, pausar, frenar, terminar la simulación, y los cambios de las entidades. Y en la sección 5.6 se hace énfasis en el requisito del seguimiento del pasajero.

5.1 - Arquitectura

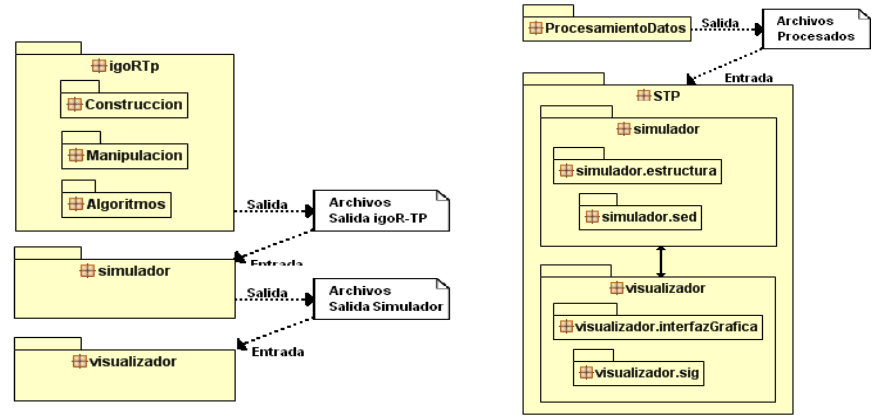
Es esta sección se describe la arquitectura desarrollada en los proyectos anteriores, y la nueva arquitectura que se diseñó para soportar los requerimientos de este proyecto [1]. En la Figura 9 se puede visualizar las diferencias entre las dos arquitecturas.

Arquitectura anterior:

La arquitectura utilizada hasta este momento tiene tres componentes independientes entre sí [6]:

1. Procesamientos de datos de entrada (IgorTP):Herramienta que permite generar casos de estudio.
2. Simulador: A partir de los datos procesados, se realiza la simulación del caso de estudio.
3. Visor: Luego de ejecutar la simulación, a partir de archivos de salida del mismo se ejecuta la visualización de lo simulado.

Además de que estos componentes son independientes entre sí, también tienen la particularidad que en una misma simulación no se puede ejecutar de forma simultánea el simulador y el visor, ya que la entrada del visor corresponde a la salida del simulador. En la Figura 9 (A) se muestra la arquitectura que se utiliza.



A) arquitectura anterior **B) nueva arquitectura**

Figura 9 - Comparación de arquitecturas

Arquitectura nueva:

La principal limitante de la arquitectura anterior, es que no permite la visualización y manipulación en tiempo real de la simulación. Lo que permitiría saber -en forma simple- el estado actual del sistema o de algún componente particular. Para poder contemplar este requerimiento, se realizan cambios a la estructura con la que se contaba, modularizando los componentes necesarios.

En la Figura 9 (B), se muestran los módulos del sistema. El primer módulo es el procesamiento de los datos para normalizarlos según las necesidades de los otros dos componentes. Luego se encuentran el simulador y visualizador de la simulación. El componente del simulador consta de dos subcomponentes: la estructura y el simulador en sí (SED). El visualizador se encarga de la interfaz gráfica y de los componentes del SIG.

5.2 - Diseño e implementación del simulador

En esta sección se describe cómo fue desarrollado el componente del simulador. El mismo consta de una estructura y un módulo de SED. A su vez por más que pertenezca a la estructura, se dedica una subsección especial para explicar el funcionamiento e implementación de las estrategias, y cómo poder agregar nuevos comportamientos.

La diferencia de los proyectos anteriores, es que la estructura y el modelo de simulación fueron desarrollados como uno solo, mientras que en el nuevo sistema se desarrollan en dos componentes diferentes. Esto fue realizado de esta manera para que posibles modificaciones futuras en la estructura produzcan menor impacto en el módulo de SED. En contrapartida a este beneficio, los pasajeros y los ómnibus aparecen en ambos componentes. Por lo que hay dos entidades para cada uno de ellos y se relacionan con el atributo identificador, que debe ser el mismo en ambos componentes

5.2.1 - Estructura

La base para todo el desarrollo del sistema, es la estructura. A través de ella es que funciona el simulador, y es el nexo para realizar la representación visual tanto de los pasajeros como de los ómnibus. A continuación se detalla la estructura del proyecto *"Simulación de sistemas de transporte público con servicios de información a usuarios en tiempo real"* -que se tomó como idea de base- para luego describir las modificaciones realizadas [1].

Diseño Estructura Anterior

En la Figura 10 se muestra el diseño de la estructura base que se utiliza en los proyectos anteriores, y que también se utiliza como referencia para este proyecto.

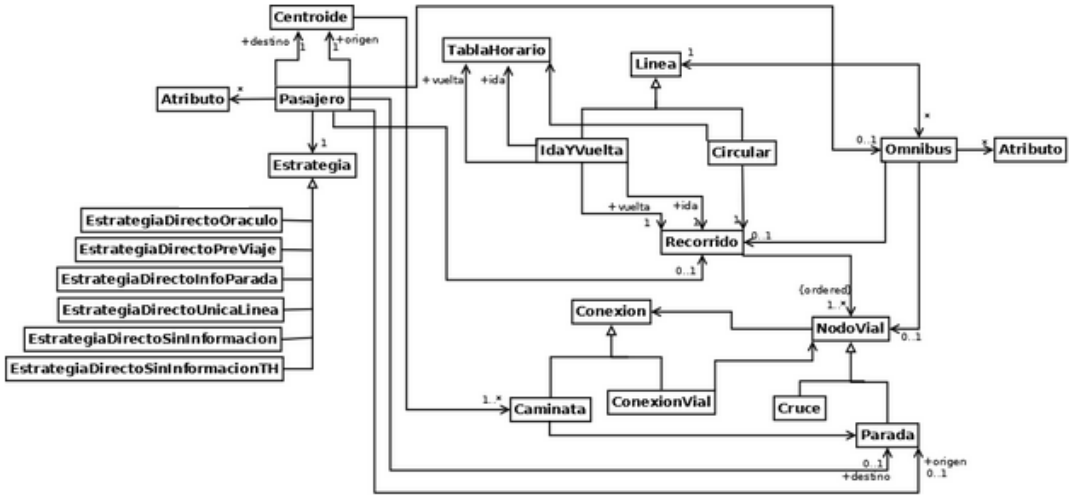


Figura 10 - Diagrama de clases de la estructura del proyecto anterior

A continuación se realiza una breve descripción de las clases del diagrama:

- **Centroide:** La clase modela un punto ficticio que representan zonas geográficas concentradas.
- **NodoVial:** La clase modela un nodo vial, que puede ser tanto un cruce o una parada. Es una clase abstracta, que contiene las coordenadas físicas para poder calcular las distancias entre todos los nodos viales.
- **Cruce:** La clase modela un cruce (intersección de calles). Es una implementación de NodoVial.
- **Parada:** La clase modela una parada de ómnibus. Es una implementación de NodoVial.
- **Caminata:** La clase modela la caminata de un pasajero. Puede ser desde un centroide a una parada, de una parada hacia otra o desde una parada a un centroide. Tiene como atributo, aparte del identificador, un tiempo en que se demora en hacer esa caminata.
- **ConexionVial:** La clase modela una conexión vial entre dos nodos. Tiene asociado como costo, el tiempo promedio en que un ómnibus demora en recorrer la conexión.
- **Linea:** La clase modela una línea de ómnibus. Las líneas de ómnibus pueden diferenciarse según su recorrido, según sea en un sentido de ida y otro de vuelta, o bien circular.
- **IdaYVuelta:** La clase modela una línea con recorrido de ida y vuelta.
- **Circular:** La clase modela una línea con recorrido circular.
- **Omnibus:** La clase modela un ómnibus.
- **Recorrido:** La clase modela un recorrido de una línea, representado por una lista de nodos viales ordenados.
- **Estrategia:** Entidad base. Provee dos operaciones abstractas, las cuales diferencian la resolución del problema según las posibilidades de acceso a información (y ponderación de la misma) por parte del pasajero.
- **Pasajero:** Representa a un pasajero en el sistema.

Diseño Nueva Estructura

La estructura que soporta el simulador, se puede visualizar en la Figura 11 . Como se puede comparar entre las figuras, la estructura es parecida.

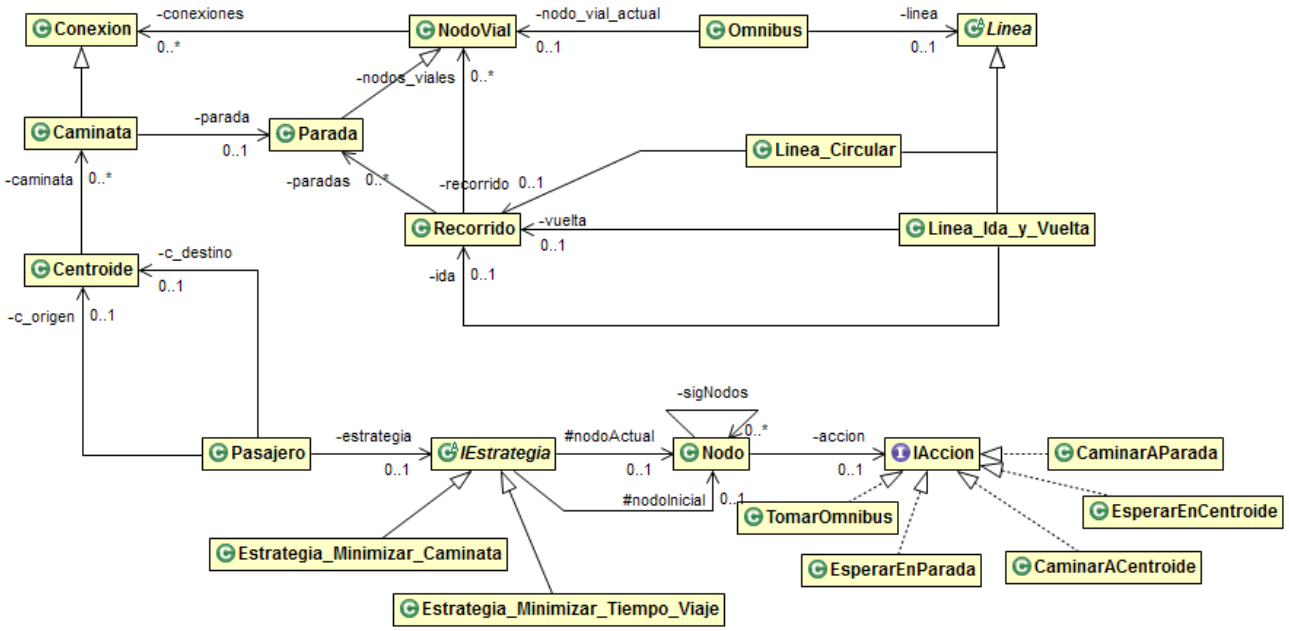


Figura 11 - Diagrama de clases de la nueva estructura

A continuación se detallan los cambios realizados.

Modificaciones en el sistema

En el sistema anterior, se tomaba en cuenta un grafo de la ciudad para el cálculo de las distancias y tiempos tanto de la flota de ómnibus como las caminatas de los pasajeros. El grafo estaba formado por cruces y paradas como los nodos y la unión de estos como las aristas.

En el nuevo sistema se optó por no modelar el grafo ya que, para el simulador, sólo se necesitan los tiempos de demoras de las caminatas y los tiempos entre paradas de los ómnibus. Por lo que los cruces no son tenidos en cuenta y por lo tanto un recorrido pasa a ser un conjunto ordenado de paradas. Las caminatas pasan a tener la distancia del recorrido y no el tiempo que se demoran en hacerla. Así el tiempo en que se hace una caminata depende de la velocidad del pasajero. Este es un cambio importante con respecto al funcionamiento anterior, ya que los ómnibus en el simulador solo van de parada a parada y no pasa por todos los nodos del grafo de una ciudad. Esto hace que el tiempo de procesamiento sea menor a la hora de simular. La representación visual de los recorridos reales de los ómnibus es responsabilidad del visualizador. La tabla de horarios que antes estaba como una entidad aparte, se deja como propiedad de la entidad Línea.

Otro de los puntos importantes de diferencia entre los sistemas, es la selección de estrategias. En el sistema anterior -si bien el diseño soporta diferentes estrategias- utilizaba siempre la misma estrategia para todos los pasajeros. En el nuevo sistema dos pasajeros pueden tener distintas estrategias en una misma ejecución. Además se modifica la forma en que se realizan la estrategias, dejando a la implementación de cada una la libertad de realizar el recorrido que crea conveniente el pasajero durante el viaje. La siguiente subsección (*5.2.2*) entra en detalle del funcionamiento de la creación, actualización y selección de las estrategias.

Controlador y Manejadores de la estructura

Un detalle no menor a la hora de la implementación del sistema, es la forma de acceder a la estructura.
La clase EstructuraImpl funciona como el controlador de la estructura y deriva a los manejadores las funcionalidades que le corresponden a cada uno, como se muestra en la Figura 12 .

A continuación se realiza una breve descripción de las clases del diagrama:

MNodoVial: Es el manejador de las paradas. Tiene las funciones básicas de agregar, quitar y obtener por identificador las paradas.

MCentoides: Es el manejador de los centroides. Aparte de las funciones básicas, se puede a partir de una parada obtener el centroide correspondiente a la caminata.

MLineas: Este manejador controla las líneas de ómnibus, sus recorridos y algunos atributos que son iguales para todos los ómnibus de esta línea, como los son: la velocidad media y su desviación estándar, la capacidad de los ómnibus y el tiempo en que esperan en una parada. Además de las funciones básicas para sus atributos, a partir de un recorrido se puede saber a cuál línea pertenece.

MOmnibus: Es el manejador de los ómnibus. Tiene las funciones básicas de agregar, quitar y obtener por identificador los ómnibus.

MPasajeros: Es el manejador de los pasajeros. Tiene las funciones básicas de agregar, quitar y obtener por identificador. También contiene los valores para velocidad media y desviación estándar para calcular la velocidad de cada pasajero. Los demás atributos son utilizados en el seguimiento de pasajeros, los cuales serán explicados en la sección *5.6 - Seguimiento de Pasajeros*, de este capítulo.

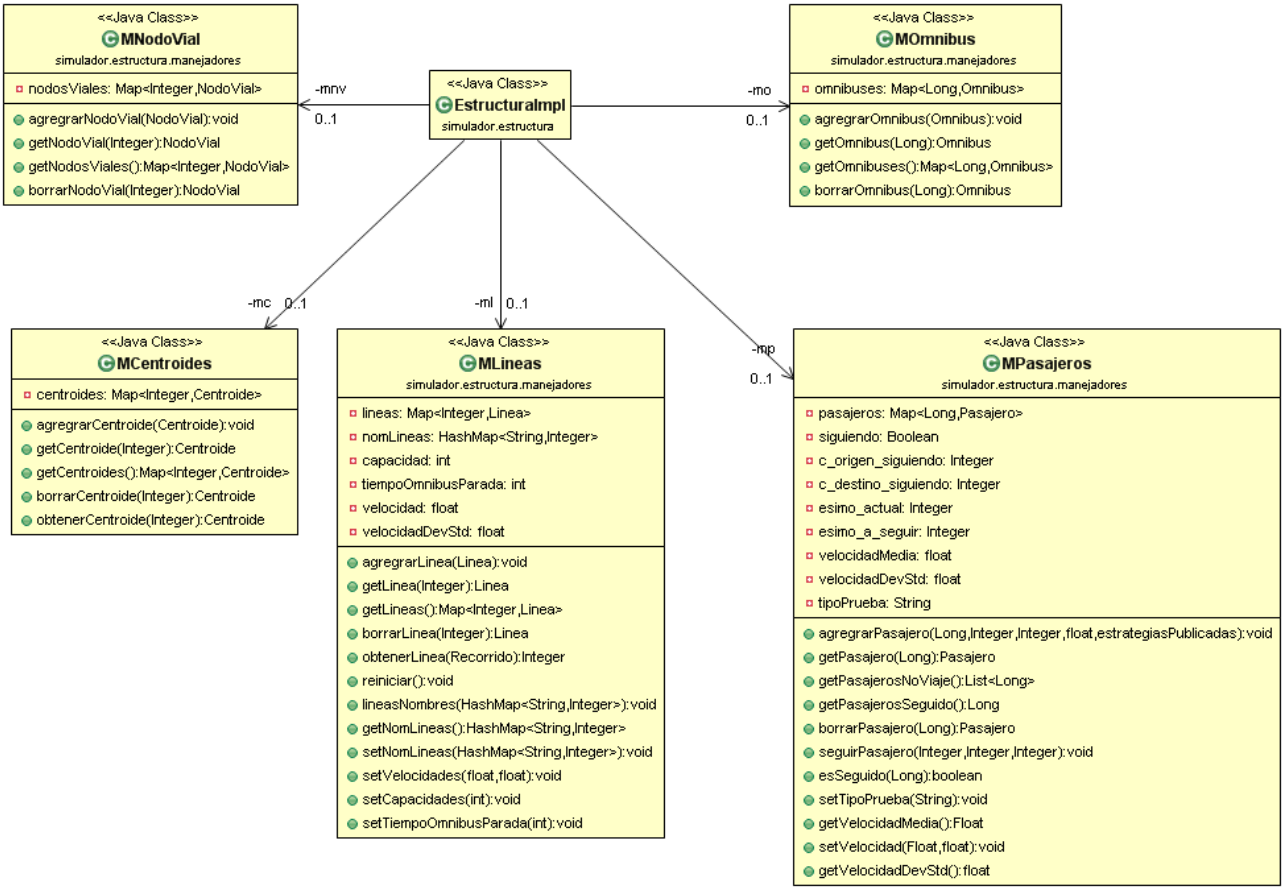


Figura 12 - Diagrama de clases para el acceso a la estructura

5.2.2 - Estrategias

Cuando un pasajero tiene que realizar un viaje desde un centroide a otro, necesita tener un plan de viaje e ir decidiendo cuál camino realizar. Las estrategias son ese plan de viaje con las decisiones que debe tomar. A partir de ellas, se genera una serie de acciones ordenadas a cumplir, para llegar del centroide origen al centroide destino. El diseño de las estrategias se puede ver en la Figura 13 .

Un pasajero siempre tiene asociada una estrategia. Para el alcance de este proyecto, esa estrategia será siempre la misma durante todo el viaje. Su implementación consta de una clase abstracta *IEstrategia*, la cual tiene asociado un nodo inicial (de tipo Nodo) que es la raíz de un árbol de nodos. Cada Nodo tiene una acción (*IAccion*) a realizar. Cada una de las ramas de este árbol, representa un posible viaje a realizar por el pasajero desde su origen a su destino.

Las posibles acciones que implementan *IAccion*, a realizar son:

EsperarEnCentroide: La acción indica el centroide y el tiempo que el pasajero debe esperar en él. Cabe aclarar que la espera en centroide se puede dar únicamente en el centroide origen del recorrido del pasajero, ya que cuando éste arribe al centroide destino, culmina su ciclo de vida.

CaminarAParada: Indica a qué parada debe caminar un pasajero y en qué instante de tiempo hacerlo. Si bien el tiempo de caminata puede ser cualquiera, normalmente se calcula como la distancia de la caminata sobre la velocidad del pasajero.

EsperarEnParada: Indica la parada en la que se encuentre un pasajero, la acción indica el tiempo que el pasajero debe esperar en ella. Esta acción es de utilidad para casos en los que un pasajero no debe tomar el próximo ómnibus de su conveniencia que arribe a la parada, por ejemplo para casos en que tenga que llegar a destino a cierta hora y desee hacer tiempo en la parada. Mientras esta acción se esté ejecutando, el pasajero no está esperando el arribo de ningún ómnibus, está ocioso esperando el tiempo establecido.

TomarOmnibus: Indica la línea, el tipo de recorrido que debe tomar un pasajero, y la parada en la que debe descender para poder llegar al destino, ya sea para poder caminar al centroide destino como para continuar con su viaje.
CaminarACentroide: Indica a qué centroide debe caminar el pasajero y en qué instante de tiempo debe hacerlo. Si bien el tiempo de caminata puede ser cualquiera, normalmente se calcula como la distancia de la caminata sobre la velocidad del pasajero. Esta debe ser siempre la última acción a realizar y no puede estar en medio de un recorrido. De no cumplirse esta restricción, no se respetaría el ciclo de vida del pasajero.

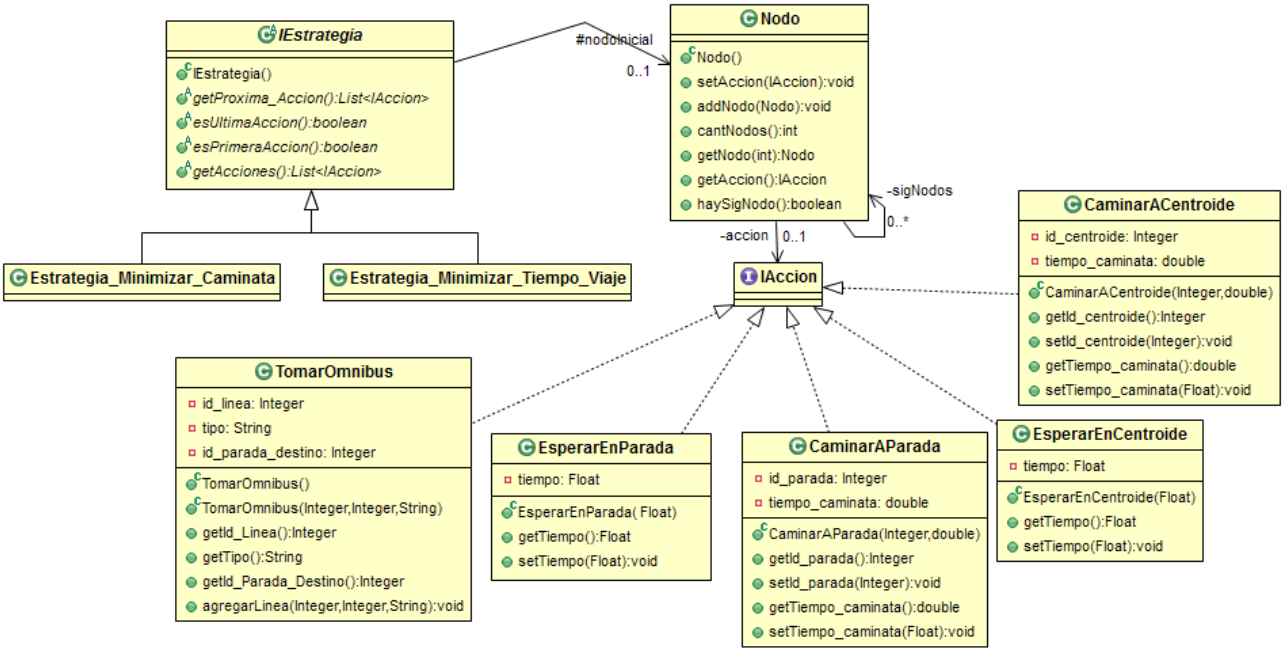


Figura 13 - Diseño de las estrategias

Funcionamiento de las estrategias

Cuando se crea un pasajero, se le asigna una estrategia a realizar, y esta genera un árbol de los posibles caminos para el par <origen, destino>, según su cometido. En la Figura 14 se muestra un ejemplo de un árbol generado por una estrategia.

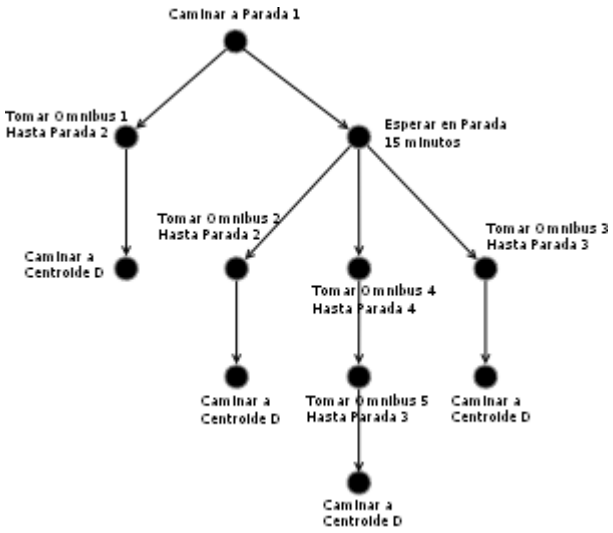


Figura 14 - Árbol generado por una estrategia

A medida que se avanza en la simulación, el pasajero va solicitando a la estrategia cuál es la siguiente acción a realizar a través de la operación *getProxima_Accion()*, que retorna una lista de acciones *(IAccion*). La lista de *IAccion* contiene todas las posibles acciones a realizar en ese momento y el pasajero hará la primera que ocurra. Por ejemplo: Un pasajero que llega a una parada puede esperar un tiempo en la parada o tomar determinada línea de ómnibus.

- Si lo primero que pasa es el término de un tiempo de espera, la estrategia seguirá por la rama correspondiente a esta acción.
- Si lo primero que pasa es la línea que estaba esperando, seguirá la rama correspondiente a esta acción.

Si bien por el alcance de este proyecto, no se llega a generar una estrategia concreta que recalcule el plan a medida que avanza la simulación, sí se contempla esta funcionalidad. La operación *getProxima_Accion(),* no solo sirve para devolver la lista de acciones a realizar, sino que en ella se puede recalcular la estrategia. Por ejemplo, si la estrategia obtiene información de los ómnibus en tiempo real, cada vez que necesita realizar una acción va a tener información que antes no tenía, por lo que es muy posible que necesite generar un nuevo plan.

Generación de nuevas estrategias

En este proyecto se generaron dos estrategias concretas: Minimizar tiempo de viaje maximizando el tiempo de espera en el centroide origen; y Minimizar distancias de caminatas. Con estas dos estrategias se pueden realizar estudios para validar el modelo y verificar que la solución desarrollada sea adecuada para los requerimientos solicitados. Es de interés para trabajos posteriores, la generación de nuevas estrategias que contemplen distintos aspectos de la conducta de los pasajeros. Para poder generar y utilizar una estrategia se debe realizar lo siguiente:

1. Implementar la nueva clase en el package *"simulador.estructura.estrategias"*, extendiendo la clase abstracta *IEstrategia*.
2. En ese mismo package existe el enumerado *estrategiasPublicadas*. En ese enumerado se registran las estrategias que se pueden utilizar. Ingresando el nombre de la estrategia implementada en el enumerado, queda disponible para poder utilizarse.

Luego de generadas y publicadas las estrategias, cuando a un pasajero se le asigne una estrategia, se crea una instancia de la clase a través de Java Reflection[1].

A continuación se detallarán las implementaciones de las estrategias concretas desarrolladas en este proyecto.

Implementación de estrategias concretas

Las estrategias concretas implementadas, dado el centroide origen y el centroide destino de un pasajero, comienzan verificando cuales son los recorridos de línea que satisfacen las necesidades del pasajero.

Para ambas estrategias (para elegir un viaje a realizar) en primera instancia se obtienen todos los recorridos que pasan por una parada (inicial), la cual tiene una caminata que relaciona esa parada con el centroide origen. Y en segunda instancia, para cada uno de estos recorridos, controla que exista una parada del recorrido posterior a la inicial, que se relacione a través de una caminata con el centroide destino. El proceso es el siguiente:

1. Obtener todas las Paradas de Caminatas que tiene como Centroide el centroide origen.
2. Obtener todos los recorridos que pasan por las Paradas obtenidas en el punto anterior.
3. Obtener todas las Paradas de Caminatas que tienen como Centroide el centroide destino.
4. Obtener todos los recorridos que pasan por las Paradas obtenidas en punto anterior.
5. Interceptar los dos conjuntos obtenidos en 2 y 4.
6. Corroborar que en los recorridos obtenidos la parada inicial esté antes de la parada final.

1 Java Reflection es una funcionalidad que puede inspeccionar y manipular clases e interfaces (así como sus métodos y campos) en tiempo de ejecución.

Con esto se obtienen todos los recorridos que le sirven al pasajero de forma directa (ya que no se realizan trasbordos para estas estrategias).

A partir de estos recorridos se aplicarán los filtros necesarios para seleccionar alguno de estos según la estrategia.

Minimizar tiempo de viaje maximizando tiempo de espera en centroide

En caso de que el recorrido satisfaga las necesidades del pasajero, se debe analizar si es la solución óptima. Para esto es necesario considerar los siguientes datos:

- distancia desde centroide origen a parada inicial
- distancia desde parada final a centroide destino
- parada inicial a la que debe ir el pasajero a tomar el ómnibus
- parada final en la que se debe bajar el pasajero
- distancia entre parada inicial y parada final (por el recorrido de la línea)
- distancia entre primer parada del recorrido de la línea y parada inicial del pasajero (por el recorrido de la línea)
- tabla de horarios de la línea
- tiempo de simulación
- velocidad de la línea
- velocidad de caminata del pasajero

Con estos datos particulares se deben calcular los siguientes valores:

1. tiempo que le lleva al pasajero caminar desde el centroide origen a la parada inicial
2. tiempo que le lleva al pasajero caminar desde la parada final al centroide destino
3. tiempo de recorrido del pasajero en el ómnibus
4. tiempo que le lleva al ómnibus arribar a la parada inicial del pasajero
5. máxima espera en centroide que puede realizar el pasajero sin que arribe el ómnibus, teniendo en cuenta tiempo de simulación, 4 y 1

Considerando estos valores, se busca maximizar la función objetivo eligiendo la línea que cumpla que la sumatoria de 5, 3, 2 sea la menor de todas las combinaciones; en caso de que dos combinaciones coincidan se elige la que tenga mayor espera (valor en el punto 5).

A partir de la combinación óptima se instancia la estrategia usando las acciones EsperarEnCentroide (en caso de existir espera), CaminarAParada, TomarOmnibus, CaminarACentroide.

Minimizar distancias de Caminatas

En caso de que el recorrido satisfaga las necesidades del pasajero, se debe analizar si es la solución óptima. Para esto es necesario considerar los siguientes datos:

- distancia desde centroide origen a parada inicial
- distancia desde parada final a centroide destino

Con estos datos particulares se calcula la caminata total, se elige la línea que cumpla que la sumatoria sea mínima. A partir de la combinación óptima se instancia la estrategia usando las acciones Caminar a Parada, Tomar Ómnibus, Caminar a Centroide.

5.2.3 - Implementación del SED - Orientado a DESMO-J

En el *Anexo 3 - Elección de simulador y pruebas funcionales*, se encuentran los motivos por el cual se decidió utilizar la librería DESMO-J para el desarrollo del modelo de simulación (componente SED). El modelado de un sistema en DESMO-J, utilizando el enfoque de eventos, se realiza implementando clases derivadas de Model, Event y Entity.

Model es una clase abstracta que contiene el modelo del sistema a simular. En este se definen las colas de espera, las entidades, los eventos y las distribuciones que rigen los aspectos aleatorios del sistema. Las operaciones que se deben implementar son:

- Constructor.
- init: en esta operación se inicializan los aspectos estáticos del modelo, se deben crear instancias de las distribuciones, las colas y las entidades globales del sistema.
- doInitialSchedules: en esta operación se realizan los primeros pasos para poder comenzar la simulación. O sea que se agendan los eventos que deben poner en funcionamiento la simulación, los eventos generadores de pasajeros y ómnibus.

Las *colas* de espera se implementan instanciando la clase **Queue**. Dado que para cada parada existen dos colas (una para modelar los ómnibus que están detenidos, otra para modelar los pasajeros esperando en ella) es necesario agruparlas en la clase Colas_Paradas_Sim. Se definen tres estructuras de tipo HashMap para manejar los distintos conjuntos de colas:

- colas_paradas (clave id_parada, valor Colas_Paradas_Sim)
- colas_pasajeros_omnibus (clave id_omnibus_sim, valor Queue<Pasajero_Sim>)

El manejo de la aleatoriedad en DesmoJ se realiza por medio de la clase **Distribution**, las instancias se definen y utilizan en Model. En la clase Distribution se encapsula el comportamiento de una distribución probabilística (real o discreta). Es la superclase de todas las distribuciones que provee DesmoJ. Las instancias de Distribution contienen un generador de números aleatorios, con el cual muestrean una variable aleatoria. Se define una estructura de tipo HashMap para manejar el conjunto de distribuciones que refiere a la velocidad de las líneas:

- dist_Velocidad_Linea (clave id_linea, valor ContDistNormal)

Cabe aclarar que esta definición permite que la velocidad de una línea varíe en cada tramo del recorrido, solicitando una nueva muestra de la distribución por tramo. Se define una estructura de tipo matriz para almacenar la distribución correspondiente a la demanda de pasajeros entre cada cada par origen-destino:

- matriz_OD_dist_Arribos_Pasajeros (matriz con cada elemento de tipo ContDistExponential)

Además se definen las siguientes distribuciones, para manejar la velocidad de caminata de los pasajeros y la aleatoriedad en cuanto a la estrategia que toman los pasajeros:

- dist_Velocidad_Caminata (ContDistNormal)
- dist_uniform (ContDistUniform)

Los histogramas definidos son de tipo **Tally**, estos son:

- tiempo_Pasajero_En_Sistema (última replicación)
- tiempoMedioEnSistema (acumulado de todas las replicaciones)

Las entidades modeladas instancian la clase **Entity**, es la superclase de todas las entidades del sistema. Se debe implementar un constructor y manejo de los atributos de la entidad (en caso de que tenga). Las dos entidades definidas son Omnibus_Sim y Pasajero_sim. Tanto las entidades que refieren a los Ómnibus como las que refieren a los Pasajeros tienen un identificador único dentro de su tipo de entidad.

La clase Pasajero_Sim, aparte del atributo identificador consta con una variable de tipo TimeInstant para registrar cuando se crea el pasajero en el modelo. Este valor es usado para el histograma tiempo de viaje de pasajeros.

Event es la superclase de los eventos de un sistema. En DESMO-J se definen 2 tipos de eventos:

- Internos al sistema: son aquellos que necesitan de una entidad para ser invocados. Este tipo de eventos heredan directamente de Event. Para este tipo de eventos se debe implementar la operación abstracta eventRoutine(Entity), que define las acciones que se toman en el evento en cuestión.
- Externos al sistema: son aquellos que definen acciones externas al sistema. Son los eventos que generan los inputs del sistema. Estos heredan de la clase ExternalEvent. En estos eventos se debe implementar la operación abstracta eventRoutine que es equivalente al descrito anteriormente, solo que no necesita de una entidad para ser invocado.

Los eventos externos al sistema en nuestro caso son los generadores de las entidades Omnibus_Sim y Pasajero_sim:

- Evento_Generador_Omnibus extends ExternalEvent
- Evento_Generador_Pasajero extends ExternalEvent

Los eventos internos al sistema definidos para cambiar un solo estado interno de la entidad instancian la clase Event. Para eventos que cambian el estado de dos entidades se instancia la clase EventOf2Entities.

En nuestro caso, los eventos de arribo a parada o centroide y de retiro de parada instancian la clase Event, estos eventos son:

- Evento_Arribo_Omnibus_Parada extends Event<Omnibus_Sim>
- Evento_Arribo_Pasajero_Centroide extends Event<Pasajero_Sim>
- Evento_Arribo_Pasajero_Parada extends Event<Pasajero_Sim>
- Evento_Retiro_Parada_Omnibus extends Event<Omnibus_Sim>

Los eventos de interacción entre pasajeros y ómnibus, instancian la clase EventOf2Entities:

- Evento_Fin_Bajada_de_Omnibus extends EventOf2Entities<Omnibus_Sim,Pasajero_Sim>
- Evento_Fin_Subida_Omnibus extends EventOf2Entities<Omnibus_Sim,Pasajero_Sim>

La ejecución de la simulación se implementa en la clase EjecucionSimulacion, para correr una simulación se necesitan dos clases, Model y Experiment. En el modelo se encapsula el comportamiento del sistema a simular. El experimento es la clase que proporciona la infraestructura para ejecutar la simulación de un modelo. Contiene todas las estructuras de datos necesarias para simular el modelo y se encarga de todas las salidas necesarias. Para ejecutar un experimento, se debe crear una nueva instancia de la clase experimento y una nueva instancia del modelo. Se conecta el modelo con el experimento por medio de la operación conectToExperiment. Luego de que se hallan configurados parámetros del experimento, y que este se encuentre conectado al modelo, mediante la función start del experimento se solicita que comience la simulación. Cuando se cumplan la cantidad de replicaciones solicitadas, se finaliza el experimento mediante la llamada a la función finish.

Otra consideración de DesmoJ, es que maneja el tiempo con la clases TimeInstant y TimeSpan. TimeInstant representa puntos en el tiempo de simulación, en cambio TimeSpan representan lapsos de tiempo. Todas las operaciones que impliquen conocer o asignar tiempo obtienen o crean un objeto de estos tipos con el valor deseado.

Pseudocódigo de los eventos

Según la definición de los eventos vistos en la sección *3.2 - Funcionalidades de la aplicación*, se implementan de la siguiente manera. Se asume que las estrategias generan correctamente las acciones para todos los eventos de pasajeros. En caso contrario, se agenda nuevamente el evento un minuto más tarde con el objetivo de poder recalcular el camino y continuar con la correcta ejecución del sistema. Estas acciones no se incluyen en los pseudocódigos.

Evento_Generador_Omnibus (linea l, tipo de recorrido tr) {

crear ómnibus ‘o’ en simulador
crear cola de pasajeros para ‘o‘ en simulador
crear ómnibus en estructura informado identificador de ‘o‘, ‘l‘, ‘tr‘
crear evento Evento_Arribo_Omnibus_Parada(‘o‘)
agendar Evento_Arribo_Omnibus_Parada en tiempo 0
obtener tiempo ‘t‘ de siguiente horario de salida para la ‘l‘ y ‘tr‘
agendar Evento_Generador_Omnibus(‘l‘, ‘tr‘) en tiempo ‘t‘

}

Evento_Arribo_Omnibus_Parada (ómnibus o) {

obtener de estructura identificador de próxima parada ‘pp‘ para ‘o‘
actualizar en estructura cantidad de paradas recorridas para ‘o‘
obtener de estructura tiempo de retiro de parada ‘t‘ para ‘o‘
obtener lista de pasajeros a bajar ‘pasajeros_a_bajar‘ para ‘pp‘ y ‘o‘

```
    si (hay pasajeros para bajar en la parada) {
                indicar que ‘o‘ se detuvo en ‘pp‘
                insertar ‘o’ en la cola de ómnibus de la parada ‘pp’
                para cada pasajero ‘p’ en ‘pasajeros_a_bajar’ {
                        decrementar cantidad de pasajeros de ‘o’ en estructura
                        crear evento Evento_Fin_Bajada_de_Omnibus(‘o’, ‘p’)
                        agendar Evento_Fin_Bajada_de_Omnibus en tiempo ‘t’
                }
    }
    obtener de la estructura cantidad de pasajeros de ‘o’
    obtener lista de pasajeros a subir ‘pasajeros_a_subir‘ para ‘pp‘ y ‘o‘
    si ( ‘o’ tiene capacidad disponible y hay pasajeros a subir){
        si (‘o’ no se detuvo en ‘pp’) {
                        insertar o en la cola de ómnibus de la parada pp
                        indicar que ‘o‘ se detuvo en ‘pp‘
        }
        para cada ( pasajero ‘p’ en ‘pasajeros_a_subir’ y ‘o’ tiene capacidad disponible) {
                incrementar cantidad de pasajeros de ‘o’ en estructura
                quitar ‘p’ de la cola de pasajeros de la parada ‘pp’
                crear evento Evento_Fin_Subida_de_Omnibus (‘p’, ‘o’ )
                agendar Evento_Fin_Subida_de_Omnibus en tiempo ‘t’
        }
        }
    crear evento Evento_Retiro_Parada_Omnibus(‘o’, ‘paro_en_parada’)
    agendar Evento_Retiro_Parada_Omnibus en tiempo ‘t’
}
```

Evento_Fin_Subida_de_Omnibus(ómnibus o, pasajero p){

```
    actualizar en estructura estado “En viaje” de pasajero ‘p’
    actualizar en estructura parada destino de ‘p’ dado que tomó ‘o’
    insertar ‘p’ en la cola de pasajeros de ‘o’
    quitar en estructura las líneas posibles a tomar para ‘p’
```

}

Evento_Fin_Bajada_de_Omnibus(ómnibus o, pasajero p){

```
    quitar ‘p’ de la cola de pasajeros de ‘o’
    crear evento Evento_Arribo_Pasajero_Parada(‘p’)
    agendar Evento_Arribo_Pasajero_Parada en tiempo 0 para la entidad p
```

}

Evento_Retiro_Parada_Omnibus (ómnibus o, boolean paro_en_parada) {

```
    obtener de estructura identificador de parada actual ‘pa’ para ‘o’
    si ( ‘paro_en_parada’) {
                quitar ‘o’ de la cola de ómnibus de la parada ‘pa’
    }
    obtener de estructura la línea ‘l’ de ‘o’
    si (‘pa’ no es el destino de ‘l’) {
        obtener de estructura distancia a próxima parada ‘dpp’ para ‘o’
```

```
        obtener velocidad 'v' de DistribuciónVelocidadLinea('l')
        calcular tiempo tramo 't_tramo' como 'dpp' / 'v'
        crear evento Evento_Arribo_Omnibus_Parada('o')
        agendar Evento_Arribo_Omnibus_Parada('o') en 't_tramo'
    }
    sino {
        solicitar a estructura que elimine 'o'
    }
}
```

Evento_Generador_Pasajero (centroide origen, centroide destino) {

```
    crear pasajero 'p' en simulador
    obtener estrategia 'e' de DistribuciónEstrategias()
    obtener velocidad 'v' de DistribuciónVelocidadPasajero()
    crear pasajero en estructura informado identificador de 'p', 'origen', 'destino', 'e', 'v'
    crear evento Evento_Arribo_Pasajero_Centroide('p')
    agendar Evento_Arribo_Pasajero_Centroide en tiempo 0
    obtener próximo tiempo 't' de DistribucionCrearPasajero('origen', 'destino')
    agendar Evento_Generador_Pasajero en tiempo 't'
}
```

Evento_Arribo_Pasajero_Centroide (pasajero p) {

```
    obtener de estructura lista de próximas posibles acciones a realizar por 'p'
    si (hay posibles acciones) {
      // pasajero llega a centroide origen
      para cada posible acción{
            si (acción indica esperar en centroide) {
                actualizar en estructura estado "Esperando" para 'p'
                obtener tiempo de espera 'tiempo_espera' de la acción
                crear evento Evento_Arribo_Pasajero_Centroide('p')
                agendar Evento_Arribo_Pasajero_Centroide en tiempo 'tiempo_espera'
            }
                si (acción indica caminar a parada) {
                    actualizar en estructura estado "Caminando" para 'p'
                    obtener tiempo de caminata 't_caminata' de la acción
                    crear evento Evento_Arribo_Pasajero_Parada('p')
                    agendar Evento_Arribo_Pasajero_Parada en 't_caminata'
                }
            }
    }
    sino {
      // pasajero llega a centroide destino
      solicitar a estructura que elimine 'p'
      loguear el tiempo de viaje del pasajero en histograma, dado tiempo actual y tiempo que inició en el
sistema
    }
}
```

```
Evento_Arribo_Pasajero_Parada (pasajero p) {
    informar a estructura de que 'p' arribó a parada 'pa'
    obtener de estructura lista de próximas posibles acciones a realizar por 'p'
    si (hay posibles acciones) {
        para cada posible acción{
            si ( acción indica esperar en parada) {
                actualizar en estructura estado "Esperando" para 'p'
                obtener tiempo de espera 't_espera' de la acción
                crear evento Evento_Arribo_Pasajero_Parada('p')
                agendar Evento_Arribo_Pasajero_Parada en tiempo 't_espera'
            }
            si (acción indica caminar a otra parada) {
              obtener parada 'pp' de la acción
              actualizar en estructura a 'pp' como la parada actual de 'p'
              actualizar en estructura estado "Caminando" para 'p'
              obtener tiempo de caminata 't_caminata' de la acción
              crear evento Evento_Arribo_Pasajero_Parada('p')
              agendar Evento_Arribo_Pasajero_Parada en 't_caminata'
            }
            si (acción indica caminar a centroide) {
              actualizar en estructura estado "Caminando" para 'p'
              obtener tiempo de caminata 't_caminata' de la acción
              crear evento Evento_Arribo_Pasajero_Centroide('p')
              agendar Evento_Arribo_Pasajero_Centroide en 't_caminata'
            }
            si (acción indica tomar ómnibus) {
              actualizar en estructura estado "Esperando" para 'p'
              obtener identificador de línea 'id_linea' de la acción
              obtener parada destino 'p_destino' de la acción
              obtener tipo de linea 'tipo_linea' de la acción
              agregar en estructura líneas posibles a tomar para 'p', dado 'id_linea', 'p_destino' y 'tipo_linea'
              insertar 'p' en la cola de pasajeros de parada 'pa'
            }
        }
    }
}
```

5.3 - Implementación del visualizador

Para la representación visual de un sistema de transporte de una ciudad es conveniente utilizar un sistema de información geográfica. Entre los motivos a decidirse por qué utilizar un SIG, se encuentra que cada vez existe mayor disponibilidad -de forma libre- de esté tipo de información. Como ya se cuenta con datos orientados a estos sistemas -shapefiles-, es más sencillo procesarlos con alguno de estos, y no desarrollar uno propio tratando de adaptar u obtener información de otra manera.

5.3.1 - Herramienta SIG - Implementación con GeoTools

Una vez decidida la utilización de un SIG, se analizaron diferentes soluciones para su implementación. En el *Anexo 2 - Elección de la herramienta SIG y pruebas funcionales*, se encuentran los motivos por los cuales

se decidió utilizar la herramienta GeoTools para el desarrollo de la aplicación y dentro de GeoTools por qué una componente con SWT[2] .

GeoTools es una biblioteca de Java de código abierto que proporciona herramientas para datos geoespaciales. En este proyecto se muestran solamente las funcionalidades utilizadas y adaptadas para el sistema desarrollado.

Implementación de GeoTools con SWT

La Figura 15 muestra una estructura básica de GeoTools con SWT.

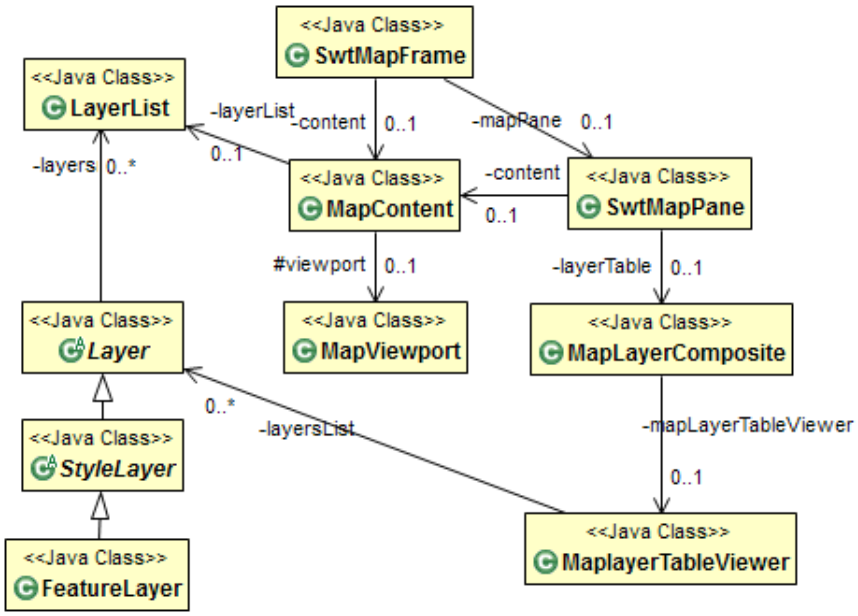


Figura 15 - Estructura básica de GeoTools con SWT

SwtMapFrame: Clase principal de la visualización. Es la encargada de mantener la coordinación entre los datos y los componentes de visualización (MapContent y SwtMapPane). Cuenta con funcionalidades sobre el mapa como lo son el Zoom In, Zoom Out, restaurar Zoom, mover el mapa y selección de elementos de capa.

MapContent: Esta clase mantiene las características de las capas del SIG para poder ser visualizada. Entre sus funciones están: agregar, eliminar, mover y ordenar las capas; mantener el sistema de referencia espacial; y mantener el área de visualización donde se dibuja el mapa.

SwtMapPane: Es la clase encargada de dibujar el contenido del MapContent en el área de visualización. Para redibujar el mapa lo hace a través de eventos, como lo es el agregar una nueva capa, acciones con el mouse, etc. Es importante aclarar en este punto que las funciones de redibujar deben ser ejecutadas por el mismo hilo de ejecución, sino carecen de efecto.

MapViewport: Hace las transformaciones de las coordenadas del mundo real (según el sistema de referencia espacial) a las coordenadas de la pantalla de la aplicación y viceversa.

MapLayerComposite: Contiene la MaplayerTableViewer y permite eliminar, ordenar, ocultar, mostrar todas las capas que se encuentran en el MapContent.

2 SWT es un toolkit de código abierto para Java diseñado para proporcionar un acceso eficaz y portátil para las instalaciones de interfaz de usuario de los sistemas operativos en los que se aplique.

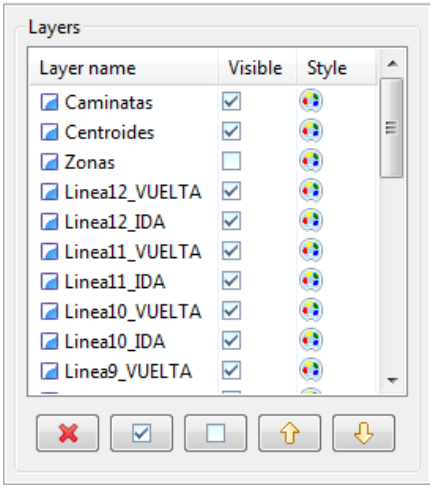


Figura 16 - Visualización del MapLayerComposite con el MaplayerTableViewer cargado

MaplayerTableViewer: Esta clase permite modificar los estilos, ocultar o visualizar cada una de las capas ingresadas en el MapContent.

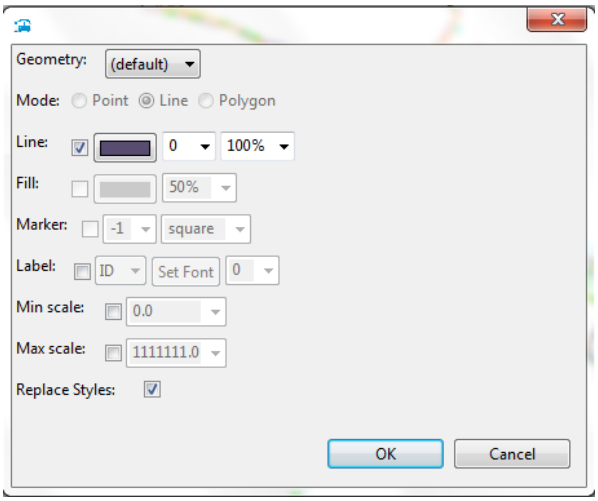


Figura 17 - Modificación de estilo de las capas del MaplayerTableViewer

LayerList: Mantiene las capas para el MapContent.
Layer: Son las capas a dibujar. Para este proyecto solo se utiliza StyleLayer.
StyleLayer: Es una clase abstracta para las capas de tipo raster y de tipo vectorial. En este proyecto se utiliza solo capas vectoriales.
FeatureLayer: Son las capas vectoriales. Contienen los elementos de la capa, y un estilo (Style) que indica cómo debe mostrarse el contenido en el mapa. A continuación se describirán los estilos utilizados en este proyecto.

Se definen los tipos de estilos según su complejidad en estilos básicos y estilos complejos.

Estilos básicos

Los estilos básicos son aquellos que tienen el mismo símbolo para cualquier estado de la capa a la que se aplica. A través de la clase SLD (Style Layer Descriptor) se pueden crear estilos para los elementos utilizados en el proyecto.

- SLD.createPointStyle(Tipo dibujo, Color borde, Color relleno, opacidad, tamaño) : Style
  Con esta función se crea un estilo para un punto.

  - Tipo dibujo: Es un String que puede ser “Circle”, “Square”, “Cross”, “X”, “Triangle” o “Star”. Por lo que la forma que toma el punto corresponde al tipo ingresado.
  - Color borde: Es el color que va a tener el borde de la figura.
  - Color relleno: Es el color que va a tener el interior de la figura.
  - opacidad: Es un valor entre 0 y 1, y define que tan translúcida es la representación visual del punto.
  - tamaño: Es el tamaño del símbolo del punto.
- SLD.createLineStyle(Color, ancho) : Style
  Con esta función se crea un estilo para una línea
  - Color: Es el color que va a tener la línea.
  - tamaño: Es el ancho de la línea.
- SLD.createPolygonStyle(Color borde, Color relleno, opacidad) : Style
  - Color borde: Es el color que va a tener el borde del polígono.
  - Color relleno: Es el color que va a tener el interior del polígono.
  - opacidad: Es un valor entre 0 y 1, y define que tan translúcido es el polígono.

Estilos complejos

Los estilos complejos son aquellos que dependiendo del estado de la capa, pueden mostrar uno u otro símbolo en el mapa. En la Figura 18 se muestra como es la diagrama de clases de los estilos.

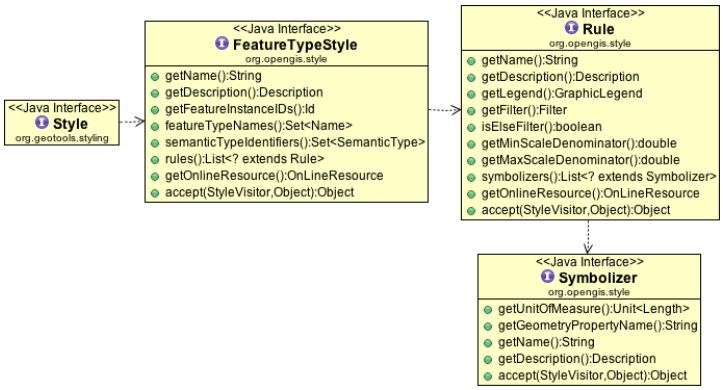


Figura 18 - Estilos complejos

Un estilo puede tener varias características. Cada una de estas características tiene una regla, que al cumplirse devuelve el símbolo asociado a la misma para dibujar en el mapa. Cada vez que hay una acción sobre el mapa es que se recalculan las reglas.

Modificaciones de GeoTools con SWT para el nuevo sistema

Con el frame utilizado por GeoTools no es posible realizar el sistema a implementar, por lo que se crea la clase SwtFrameSTP a partir de la clase SwtMapFrame, agregando las funcionalidades necesarias para cumplir con las necesidades.

La otra modificación más significativa es la creación de las clases CapaLineas, CapaPoligono, CapaPunto, Linea, Poligono y Punto para poder manipular de manera más sencilla los elementos de cada una de estas. Como las capas que mantiene GeoTools son genéricas, es más complicada su manipulación ya que pueden ser capas vectoriales o rasters, por lo que el acceso a los elementos no es trivial. De esta forma se obtiene un nivel de abstracción mayor y no tener que mediar con desarrollos más complejos cuando el sistema no lo requiere.

En la Figura 19 se muestra cómo quedó la estructura luego de las modificaciones realizadas.

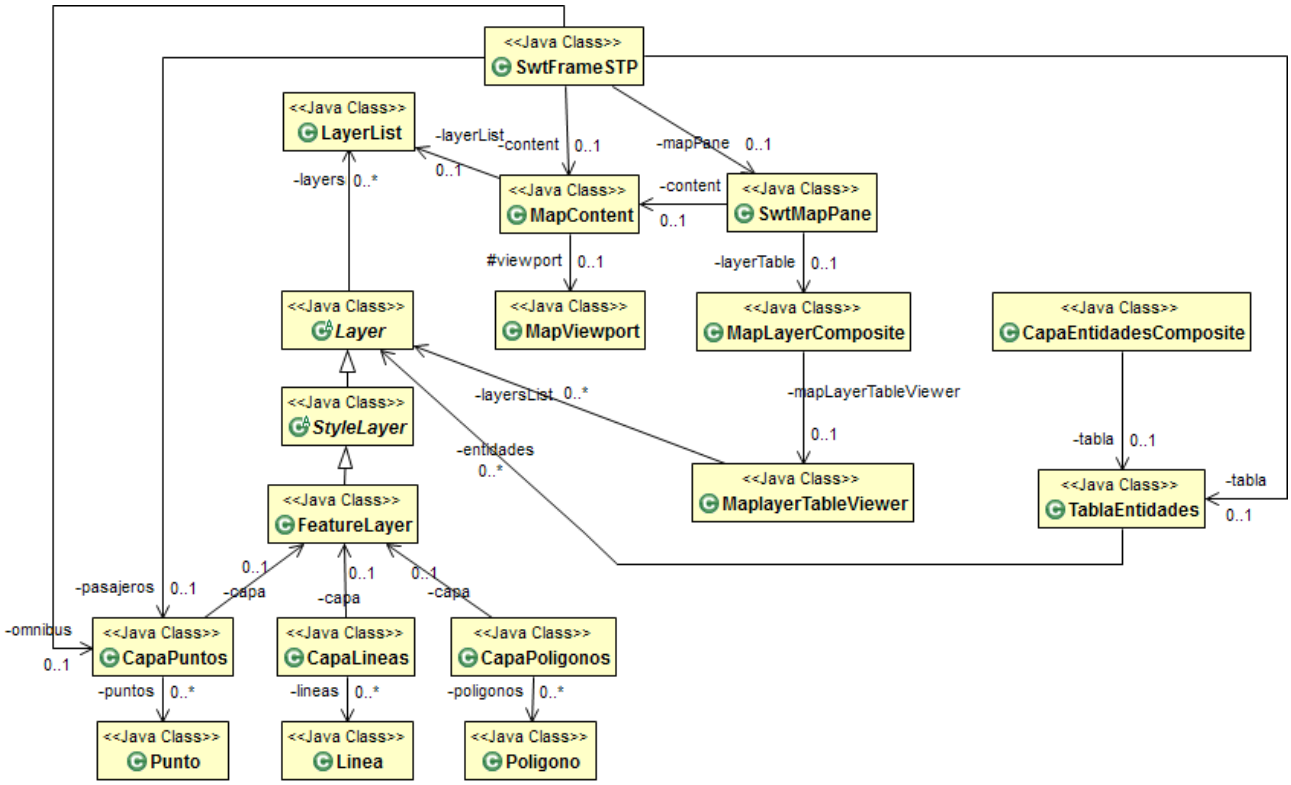


Figura 19 - Modificaciones sobre GeoTools con SWT

SwtFrameSTP: Esta clase es creada a partir de la clase *SwtMapFrame* y adaptada para las funcionalidades requeridas. Se le agregan las opciones de iniciar, pausar, continuar y frenar la simulación; seleccionar un pasajero para realizar seguimiento; mostrar el avance y el tiempo transcurrido de la simulación; modificar la velocidad de la simulación; y la visualización de elementos móviles.

CapaEntidadComposite: Se copia el funcionamiento de la clase *MapLayerComposite* pero no permite eliminar ni ordenar las elementos, solamente ocultar y mostrar todas las capas móviles. Si bien el *MapLayerComposite* se mantiene desde el *SwtMapPane*, esta clase es mantenida desde *SwtFrameSTP* para su mejor manipulación. Fue agregada para el mantenimiento de los ómnibus y los pasajeros.

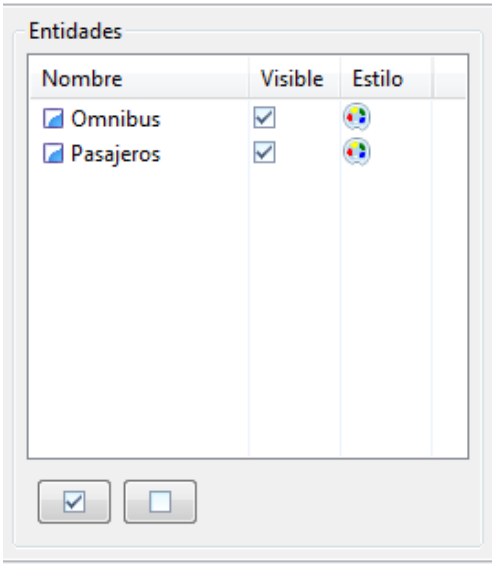


Figura 20 - Visualización del CapaEntidadComposite con la TablaEntidades cargado

TablaEntidades: Esta clase permite ocultar o visualizar las capas móviles. Además muestra el color de cada línea de ómnibus si se selecciona *"Estilo"* en la tabla de entidades. Para los pasajeros no se definió esta funcionalidad.

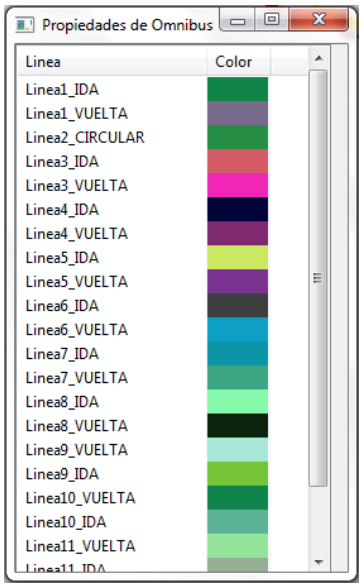


Figura 21 Color por línea de ómnibus

CapaPoligonos y Poligono: Fueron creadas para poder manipular de manera más práctica los conceptos de zonas. En sí, estas clases se utilizan para la carga del sistema (como se describe en la sección *5.4*) y para la visualización. En tiempo de ejecución, estas clases no tienen mayor funcionalidad.

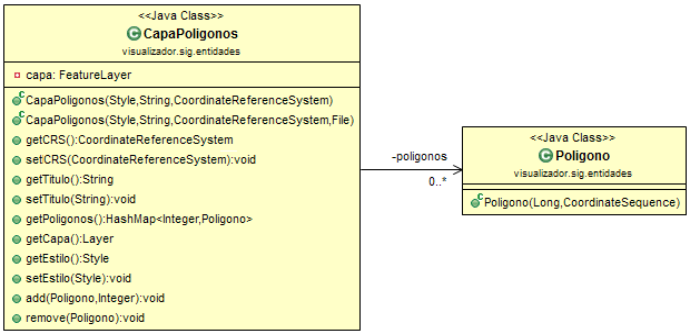


Figura 22 - Clases CapaPoligonos y Poligono

CapaPoligonos: Contiene un *FeatureLayer* (atributo *capa*) que es la que se ingresa en el *MapContent*. También tiene un hash <idPoligono, Poligono> (atributo poligonos), con todos los polígonos que se encuentran en la *FeactureLayer*. De esta forma se accede a los puntos de la capa directamente.
Hay dos maneras de crear una capa de polígonos. La primera crea una capa vacía (sin elementos en ella) con un estilo para su representación visual, un título y un CRS. La segunda crea una capa a partir de un archivo shapefile de polígonos, cargando en la capa todos los elementos que se encuentran en el archivo. Al igual que el caso anterior, se crea con un estilo para su representación, un título y un CRS.
La demás operaciones son los "getters" y "setters" de los parámetros ingresados, y poder agregar o eliminar polígonos a la capa.

Poligono: Esta clase extiende de la clase *Polygon* de GeoTools, que contiene entre otros atributos, un conjunto de coordenadas ordenadas que forman el polígono. Aparte de estos atributos se le agrega un identificador único dentro de la capa.

CapaLineas y Linea: Fueron creadas para poder manipular de manera más práctica los conceptos caminatas y líneas de ómnibus.

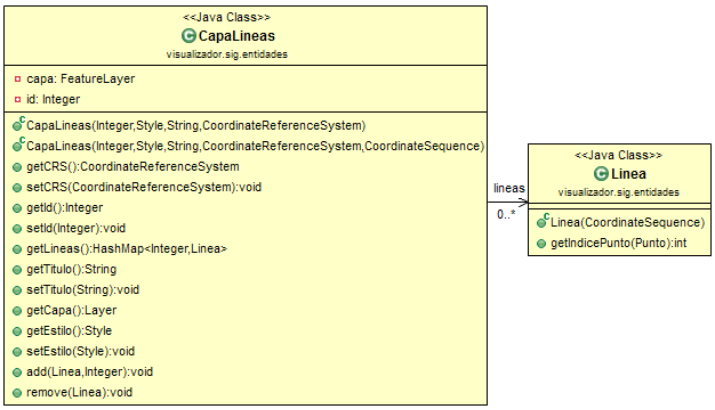


Figura 23 - Clases CapaLineas y Linea

CapaLineas: Contiene un *FeatureLayer* (atributo capa) que es la que se ingresa en el *MapContent*. El identificador id se utiliza para saber a qué recorrido de una línea de ómnibus hace referencia esta capa. También tiene un hash <idLinea, Linea> (atributo *lineas*), con todas las líneas que se encuentran en la FeatureLayer (para las líneas de ómnibus la capa tiene solo un elemento). En caso de otro tipo de línea, este valor es indiferente. Se puede crear una capa de líneas vacía, informando el identificador, el estilo de representación, el título y el sistema de referencia de la capa. Tambíen se puede crear una capa de líneas con una sola línea en ella, indicando el identificador, el estilo de representación, el título, el sistema de referencia de la capa y una secuencia ordenada de puntos por donde pasa la línea. Las demás operaciones son los “getters” y “setters” de los parámetros ingresados, y poder agregar o eliminar líneas a la capa.

Linea: Esta clase extiende a la clase *LineString* de GeoTools que a partir de una secuencia de coordenadas geográficas <x,y>, genera una línea recta entre dos puntos consecutivos de esta.
La función *getIndicePunto(punto)* devuelve el índice donde se encuentra el punto en la secuencia. En caso de no encontrar el punto devuelve -1.

CapaPuntos y Punto: Para poder manipular de forma más sencilla los ómnibus, los pasajeros, las paradas, y los centroides, se crean las clases CapaPuntos y Puntos. Hay una capa de puntos por cada uno de los grupos mencionados, y un punto para cada elemento dentro de cada capa.

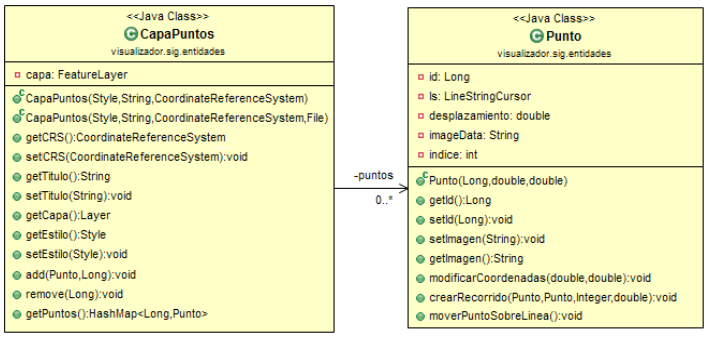


Figura 24 - Clases CapaPuntos y Punto

CapaPuntos: Contiene un *FeatureLayer* (atributo capa) que es la que se ingresa en el *MapContent*. También tiene un hash <idPunto, Punto> (atributo puntos), con todos los puntos que se encuentran en la *FeactureLayer*. De esta forma se accede a los puntos de la capa directamente.
Hay dos maneras de crear una capa de puntos. La primera crea una capa vacía (sin elementos en ella) con un estilo para su representación, un título y un CRS. La segunda crea una capa a partir de un archivo shapefile de puntos, cargando en la capa todos los puntos que se encuentran en el archivo; al igual que el caso anterior se crea con un estilo para su representación, un título y un CRS.
La demás operaciones son los “getters” y “setters” de los parámetros ingresados, y poder agregar o eliminar puntos a la capa.

Puntos: Esta clase extiende a la clase *Point* de GeoTools que contiene, entre otros atributos, el valor de las coordenadas geográficas <x,y>. Aparte de estos atributos se le agrega un identificador único dentro de la capa.
Para crear un punto se le pasa un identificador y las coordenadas geográficas <x, y>.
Los demás atributos y las operaciones sobre estos son utilizados para la representación visual de los pasajeros y los ómnibus cuando realizan un recorrido, que será explicado dentro de esta misma subsección en “Recorrido de un punto por una línea”.

Capas dinámicas y capas estáticas.
La visualización de la simulación del sistema puede ser dividida en dos grupos, los componentes estáticos (fijos) y los componentes dinámicos (móviles).

Los componentes estáticos están dados por los elementos no móviles de la estructura del sistema (como los son las paradas, los centroides y sus zonas, las caminatas y los recorridos de los ómnibus). Además se pueden agregar mapas extra (no requeridos), que pueden ser cargados por el usuario para una mejor visualización (ya sea el sistema de calles de la ciudad, un mapa de fondo, etc). Los elementos fijos son ingresados a partir de la carga de archivos, como se describe en la sección *5.4 - Carga del sistema* . La representación visual de estas capas es responsabilidad de la clase SwtMapPane. El estilo de cada capa se carga en la creación de la misma.

Los estilos utilizados para estos elementos son:

- **paradas:** Se utiliza un estilo complejo, con dos reglas. La primera es para las escalas de 0.5 a 5000, donde se muestra la representación de una parada. La segunda es para escalas de 5001 a 2000000, donde se muestra un punto de tamaño pequeño de color verde.
- **centroides:** Se utiliza un estilo complejo, con dos reglas. La primera es para las escalas de 0.5 a 5000, donde se muestra la representación de una casa. La segunda es para escalas de 5001 a 2000000, donde se muestra un punto de tamaño pequeño de color azul.
- **zonas:** Se utiliza un estilo simple de polígonos sin color de borde, con color verde como relleno y de una opacidad baja.
- **caminatas:** Se utiliza un estilo simple de línea con color anaranjado y anchura acorde a una línea.
- **recorridos:** Se utiliza un estilo simple de línea con color elegido al azar y anchura acorde a una línea.

Para mejorar la performance del visualizador, se utilizan los sub-recorridos. Estos se definen como los tramos de parada a parada de los recorridos. En “Recorrido de un punto por una línea” se describe su utilización.

Los componentes dinámicos son los pasajeros, los ómnibus, el avance y el tiempo de la simulación. Estos son creados, modificados y eliminados desde el simulador. Desde allí se le indica al visualizador qué se tiene que mostrar en cada momento. Con un hilo de ejecución que se repite cada un tiempo determinado, se redibuja cada uno de las instancias de los elementos dinámicos. El tiempo de representación se calcula como un segundo sobre la velocidad de simulación, es decir, a mayor velocidad, se redibujan los elementos

móviles más veces en el mapa.

Para representar cada instancia de los pasajeros, se obtienen todos los que no están viajando en un ómnibus en ese momento, y se dibuja según las coordenadas donde se encuentre. Un pasajero (que no está en un ómnibus) puede estar esperando en el centroide origen o en una parada, o caminando hacia una parada o al centroide destino.

Para representar cada instancia de los ómnibus se obtienen todos los del sistema, y se dibuja según las coordenadas donde se encuentre. Un ómnibus puede estar esperando en una parada o viajando de parada en parada.

Para representar los elementos móviles en el mapa, a diferencia de los elementos fijos, se procesan uno a uno. Para el caso de los ómnibus, se dibuja un punto de color correspondiente al recorrido que está realizando. Para el caso de los pasajeros, se dibuja una figura de una persona roja en caso normal y una persona azul si se le está haciendo un seguimiento a ese pasajero.

En el siguiente pseudo-código se muestra el funcionamiento del procedimiento para redibujar el mapa:

Redibujar:

```
tiempoDibujo = 1seg / velocidadSimulación;
hiloEjecucion= crearHiloEjecucion(){
                    Ejecutar() {
                        EjecutarEventoRedibujarMapa();
                        if (no terminó ejecución)
                            hiloEjecucion.ejecutarHilo(tiempoDibujo);
                        }
                };
hiloEjecucion.ejecutarHilo(tiempoDibujo);
```

Evento de representación de los elementos móviles y avance de la simulación:

```
tiempoEjecucion = ObtenerTiempoEjecucion();
avance = tiempoEjecución *100 / tiempoTotalSimulación ;
mostrarTiempoYAvanceEjecucion(tiempoEjecucion,avance );
boolean mostrarOmnibus = esVisibleCapa(Omnibus);
boolean mostrarPasajeros = esVisibleCapa(Pasajeros);
if (mostrarOmnibus) {
   para cada omnibus {
      actualizar coordenadas omnibus;
      Punto p = tarnsformarCoordMundoRealAMapa(omnibus);
        DibujarPuntoEnMapa(p);
   }
}
if (mostrarPasajeros) {
   para cada pasajero que no esté en un omnibus {
      actualizar coordenadas pasajero;
      Punto p = tarnsformarCoordMundoRealAMapa(pasajero);
      if (es pasajero en seguimiento)
          DibujarPuntoEnMapaSeguimiento(p);
      else
          DibujarPuntoEnMapa(p);
   }
}
```

Este procedimiento es realizado en el SwtFrameSTP. De esta manera se tiene mejor control de la representación de los elementos móviles y del avance de la simulación. Para los demás elementos que se encuentran en el MapContent, el control del dibujo lo tiene el SwtMapPane.

Cambio de coordenadas reales a coordenadas del frame

En el pseudo-código anterior se utiliza la función *transformarCoordMundoRealAMapa(punto)*, esto es debido a que los puntos geográficos tienen coordenadas según su sistema de referencia y la ventana donde se dibuja el mapa tiene sus propias coordenadas (que se modifican por ejemplo cuando cambia el tamaño de la ventana). Es por ello, que para poder dibujar correctamente un punto (pasajero u ómnibus), se tiene que realizar dicha transformación de las coordenadas geográficas a las coordenadas de la ventana. La clase que realiza esta acción es *AffineTransformation* y es mantenida desde el *MapContent*. Si bien la función de esta clase es más compleja, permitiendo por ejemplo modificar el sistema de referencia o realizar la función inversa, en el proyecto solo se necesitó la función mencionada.

Recorrido de un punto por una línea

La estructura no mantiene explícitamente un grafo que describa cómo es el sistema de calles. Sino que solo interesa saber para los ómnibus, cuales son las paradas por las que pasan y en qué tiempo lo hacen, y para los pasajeros cuál es el tiempo que toman en caminar de un punto a otro del sistema. Es por ese motivo que el visualizador debe ser capaz de realizar correctamente el recorrido real de un ómnibus y las caminatas de los pasajeros (aunque estas últimas son ficticias ya que son una línea recta entre dos puntos).
En la Figura 24 donde se muestra la clase *Puntos*, se pueden ver los siguientes atributos:

- **LineStringCursor ls:** Es un cursor sobre una línea a seguir por el punto. Es posible empezar al principio de la línea y a medida que se avanza, el cursor recuerda en qué punto está de la línea.
- **double desplazamiento:** Dictamina qué distancia hay que avanzar el cursor.
- **String imageData:** Contiene la dirección para dibujar una imagen que representa al punto.
- **int indice:** Es utilizado para obtener un sub recorrido (tramo de parada a parada de un recorrido).

y las siguientes operaciones:

- **setImagen y getImagen:** Manejan la posible imagen para dibujar el punto.
- **crearRecorrido(puntoInicial, puntoFinal, idRecorrido, velocidad):** Crea un cursor de un recorrido. Si no existe un Recorrido con identificador idRecorrido, el cursor creado es una línea recta entre los puntos inicial y final. Si existe el recorrido, el cursor creado es el i-ésimo sub-recorrido de este. El parámetro velocidad indica a qué velocidad debe realizar ese cursor de recorrido.
- **moverPuntoSobreLinea():** Esta función realiza el cambio de coordenadas del punto según donde se encuentra el cursor y cuanto tiene que avanzar este. Cuando se crea el cursor, se encuentra al inicio de la línea que va seguir. El cálculo del avance sobre la línea es *"velocidad/60"*. Esto es así ya que el sistema redibuja los elementos cada un segundo de sistema, y la velocidad viene dada en metros sobre minutos. De esta forma se sabe cuánto avanzó en el próximo segundo.

5.4 - Carga del sistema

Una vez finalizado el proceso de preparación de archivos (ver 3.1 - Proceso de transformación de datos), se continúa con la carga del sistema. En esta sección se describe cómo se obtiene e informa para cada componente los elementos que tiene que cargar.

En la Figura 25 , se muestra cómo es la interacción entre los componentes y subcomponentes del sistema.

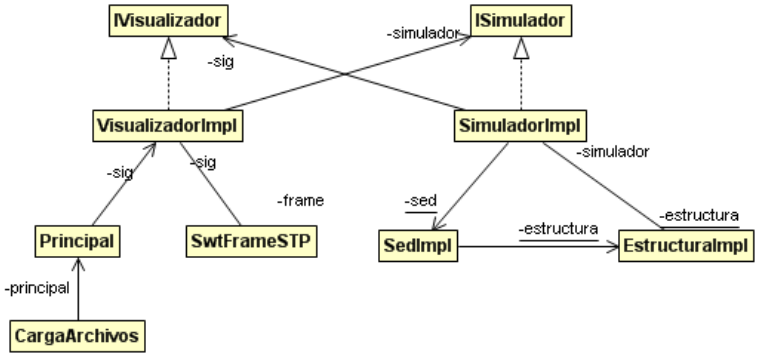


Figura 25 - Interacción entre componentes

IVisualzador: Interfaz de comunicación para el simulador.
VisualizadorImpl: Es la clase encargada de controlar todo lo referido a la visualización, desde la interfaz de usuario y la visualización del simulador. También implementa las operaciones de la interfaz *IVisualizador* y comunica la información necesaria al simulador.
Principal: Esta clase se encarga de iniciar el sistema.
CargaArchivos: Se utiliza para cargar los archivos procesados.
SwtFrameSTP: Clase que se encarga del mantenimiento y representación visual de los componentes del SIG.
ISimulador: Interfaz de comunicación para el visualizador.
SimuladorImpl: Es la clase encargada de controlar la carga de datos del simulador, implementa las operaciones de la interfaz *ISimulador* y comunica al visualizador los cambios necesarios.
EstructuraImpl: Mantiene la estructura del simulador.
SedImpl: Mantiene al simulador.

Los archivos necesarios, el orden y la forma en que se cargan son comentados a continuación. Se incluye una breve descripción de cada uno, ya que en el *Capítulo 4 - Datos de Entrada* se hace una exposición detallada de los mismos.

5.4.1 - Archivo de configuración inicial:

Este archivo es el mayormente responsable de la parametrización del sistema. No es obligatorio, ya que de no encontrarse se cargan los valores por defecto que se describen en cada parámetro.

- Velocidad media y la desviación estándar del pasajero al caminar (en km/h). En caso de que la desviación estándar sea 0, no se utiliza distribución y todos los pasajeros tienen la misma velocidad de caminata. De no existir estos valores, se cargan los valores por defecto 3,0 y 0,6 para la velocidad media y desviación estándar respectivamente.
- Velocidad media y la desviación estándar del ómnibus (en km/h). En caso de que la desviación estándar sea 0, no se utiliza distribución y todos los ómnibus tienen la misma velocidad. De no existir estos valores, se cargan los valores por defecto 10,0 y 2,0 para la velocidad media y desviación estándar respectivamente.
- Capacidad del ómnibus. Todos los ómnibus tienen la misma capacidad de pasajeros. De no existir este valor, se carga por defecto el valor 20.
- Tiempo de demora del ómnibus en la parada, si frena (en minutos). El valor por defecto es 1.
- Tiempo de la ejecución del sistema (en minutos). El valor por defecto es 120.
- Sistemas de referencia de coordenadas para las capas a utilizar. De no existir este valor, el CRS por defecto es “EPSG:32721” utilizado normalmente en Uruguay.
- Cantidad de replicaciones del experimento. Este campo es utilizado para generar varias replicacio-

nes de un experimento y así poder realizar un análisis sobre los casos de estudio. Si el valor es 1, entonces se realiza la visualización. En caso contrario, no se realiza la visualización y se ejecuta el experimento tantas veces como el valor ingresado, cambiando la semilla y obteniendo resúmenes adecuados sobre el caso de uso. De no existir este valor, por defecto es 1.

- Porcentaje de las estrategias a utilizar. Se especifica para cada estrategia el porcentaje de asignación a usuarios. Los valores por defecto son de 50% para las estrategias Minimizar Caminata y Minimizar Tiempo Viaje.
- Utilizar la distribución para la creación de pasajeros. Este valor booleano establece si el simulador debe o no utilizar la distribución Exponencial Negativa para el arribo de pasajeros. Si no se utiliza la distribución, los pasajeros son creados cada cierto tiempo fijo, dictaminado por la matriz de origen-destino. Por defecto el valor es *true*.
- Utilizar la distribución para la elección de estrategias a asignar a cada pasajero. Si el valor es true, se utiliza una distribución con diferente semilla en caso de que se ejecute varias replicaciones del caso de estudio. En caso de que el valor sea *false*, la semilla es igual para todas las replicaciones. Por defecto el valor es *true*.

Las unidades que se utilizan en el simulador son metros y minutos, por lo que el sistema hace la conversión en los casos que corresponda. El archivo no es obligatorio, ya que de no encontrarse se cargan los valores por defecto mencionados.

Implementación de la carga de los datos de configuración inicial

En la Figura 26 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

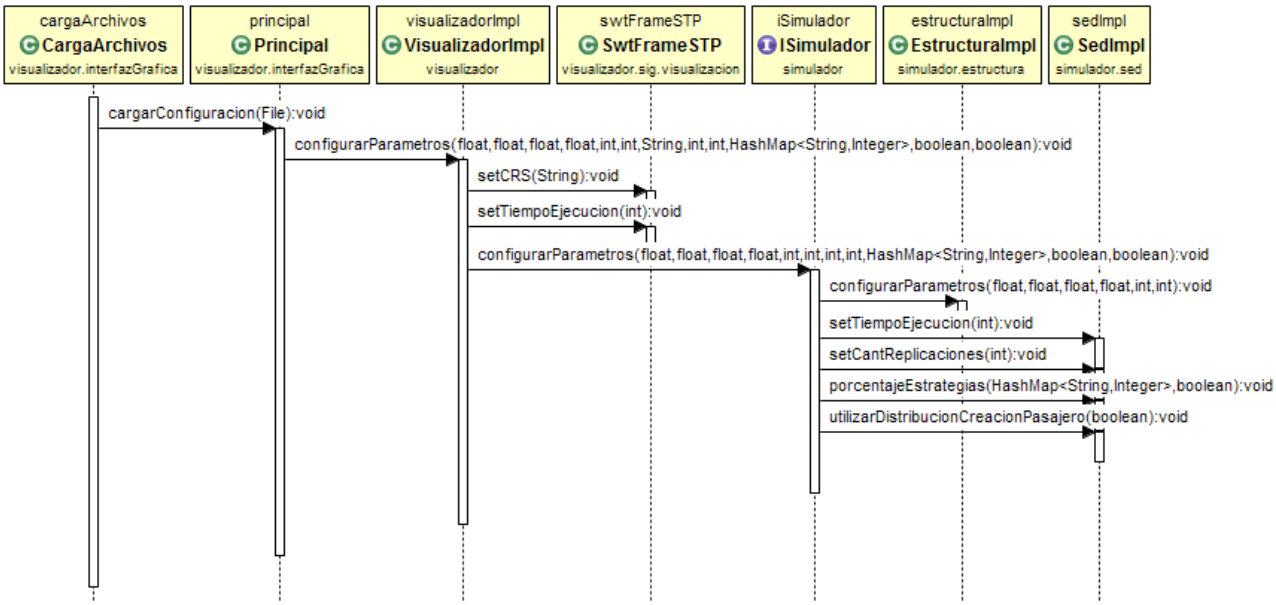


Figura 26 - Diagrama de secuencia de carga de configuración[3]

1 La clase *CargaArchivo* pasa el archivo de configuración a la clase Principal.

1.1 Cuando la clase Principal recibe el archivo de configuración, lo procesa (según la descripción anterior) para obtener los valores:

- velocidad media de persona (velMPer tipo float)
- desviación estándar de la velocidad media de la persona (velDSPer tipo float)

3 Para los diagramas de secuencia, el nivel de la numeración corresponde con el nivel de la operación que se está explicando, y el número con el número de operación dentro del nivel.

- velocidad media de ómnibus (velMBus tipo float)
- desviación estándar de la velocidad media del ómnibus (velDSBus tipo float)
- tiempo en que el ómnibus frena en la parada si tiene pasaje para subir o bajar (tParadaBus tipo int)
- capacidad de ómnibus (capacidadBus tipo int)
- el sistema de referencia espacial a utilizar (crs tipo String)
- cantidad de replicaciones (replicaciones tipo int)
- tiempo de ejecución (tEjecucion tipo int)
- porcentaje de estrategia (%Estrategia tipo HashMap<String, Integer>),
- si se utiliza distribución para la asignación de estrategias (distEstrategia tipo boolean)
- si se utiliza distribución para la creación de pasajeros (distPasajeros tipo boolean).

Una vez procesados se los pasa en ese orden a la clase *VisualizadorImpl*.

1.1.1 La clase *VisualizadorImpl* informa a *SwtFramaSTP* cual es el CRS a utilizar y este lo recuerda.

1.1.2 La clase *VisualizadorImpl* informa a *SwtFramaSTP* cual es el tiempo de ejecución total y este lo recuerda.

1.1.3 La clase *VisualizadorImpl* informa al simulador a través de su interfaz (*ISimulador*) los valores velMPer, velDSPer, velMBus, velDSBus, tParadaBus, capacidadBus, replicaciones, tEjecucion, %Estrategia, distEstrategia y distPasajeros.

1.1.3.1 SimuladorImpl (la implementación de la clase *ISimulador*) informa a la estructura mediante la clase *EstructuraImpl* los valores velMPer, velDSPer, velMBus, velDSBus, tParadaBus, capacidadBus. *EstrucutraImpl* recuerda los valores pasados.

1.1.3.2 *SimuladorImpl* informa a *SedImpl* el valor de tiempo de ejecución y este lo recuerda.

1.1.3.3 *SimuladorImpl* informa a *SedImpl* el valor de la cantidad de replicaciones y este lo recuerda.

1.1.3.4 *SimuladorImpl* informa a *SedImpl* el valor de los porcentajes de estrategias y si se utiliza una distribución para la asignación de las mismas, y este lo recuerda.

1.1.3.5 *SimuladorImpl* informa a *SedImpl* si se utiliza la distribución para la creación de pasajeros y este lo recuerda.

5.4.2 - Shapefile de paradas

Con el shapefile de paradas se crea la capa de paradas del SIG y las paradas de la estructura.
El único campo necesario para las paradas es un campo numérico único ID.

Implementación de la carga de paradas

En la Figura 27 se muestra el diagrama de secuencia de la funcionalidad y antes se presenta la explicación del diagrama.

1 La clase *CargaArchivo* pasa un archivo informando que es el shapefile de paradas a la clase *Principal*.

1.1 La clase Principal crea una clase CapaPuntos a partir del shapfile correspondiente a las paradas del sistema y se la pasa a la clase VisualizadorImpl. Este último recuerda la CapaPuntos de paradas.

1.1.1 La clase *VisualizadorImpl* envía a SwtFrameSTP la Layer (capa en GeoTools) de la CapaPuntos de paradas. SwtFrame ingresa la capa al mapa.

1.1.2 La clase VisualizadorImpl envía al ISimulador los identificadores de las paradas.

1.1.2.1 El *SimuladorImpl* envía los identificadores que llegaron al *ISimulador* a la estructura. La *EsturcutraImpl*, crea las paradas necesarias en la estructura.

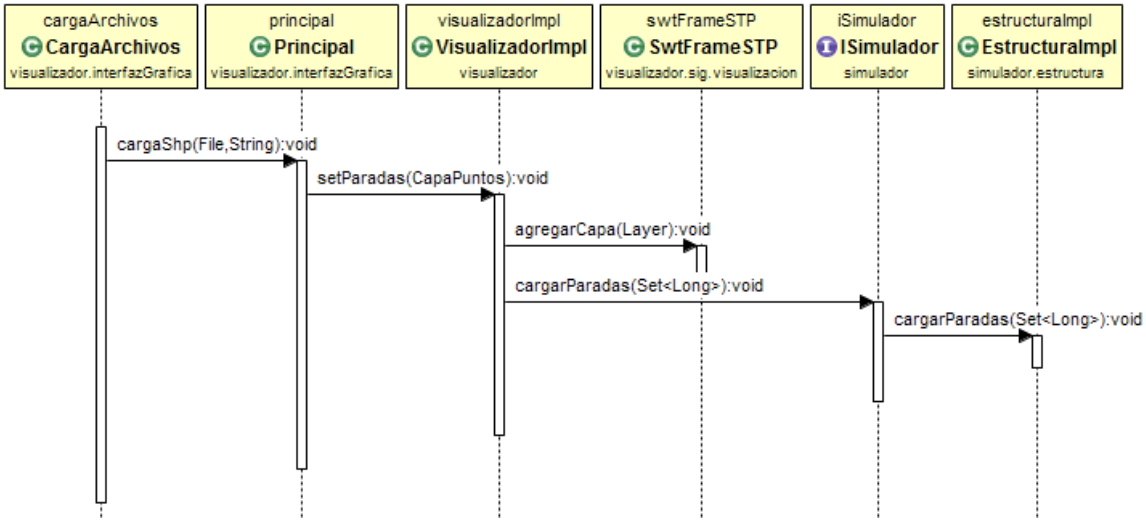


Figura 27 - Diagrama de secuencia de carga paradas

5.4.3 - Shapefile de zonas

Es el archivo generado luego del proceso de carga de datos, correspondiente a las zonas definidas como de concentración de población. A partir de este archivo se genera la capa zonas, la capa de centroides del SIG y los centroides de la estructura.

El campo ID identifica en forma única a cada zona, que es el mismo para el centroide de la zona. Los ID de las zonas tiene que ser consecutivos y empezar en cero, esto es debido a que tienen correspondencia directa con la matriz OD.

Con las paradas ya creadas, se generan las caminatas desde el centroide a todas las paradas de la zona, tanto en el SIG como en la estructura.

Implementación de la carga de zonas, centroides y caminatas

En la Figura 28 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

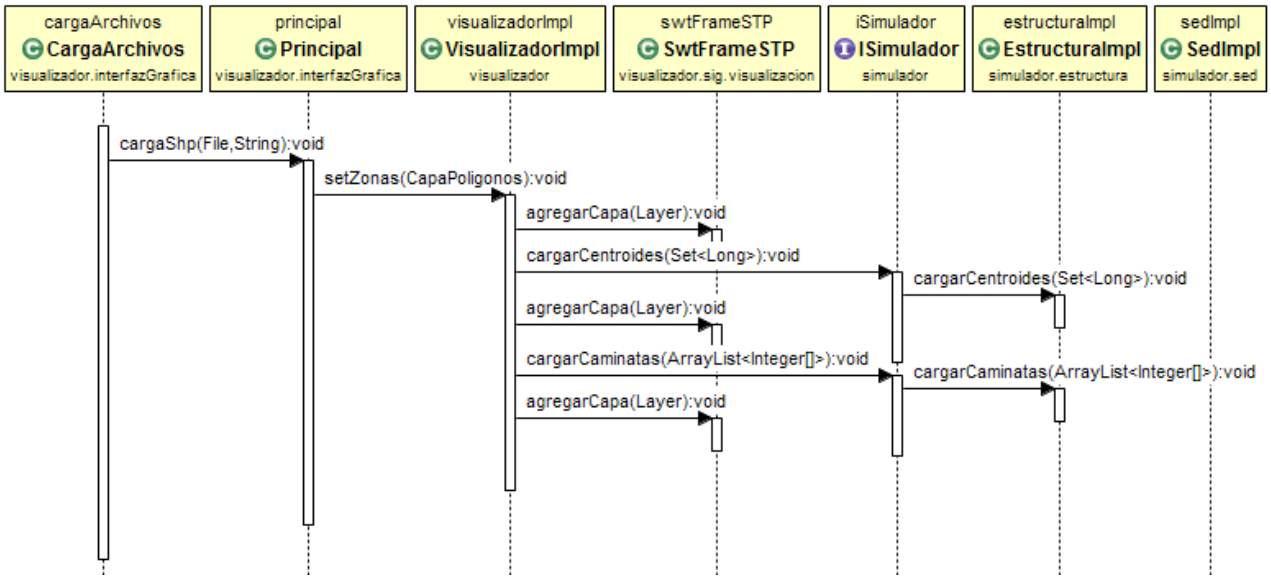


Figura 28 - Diagrama de secuencia de carga zonas, centroides y caminatas

1 La clase *CargaArchivo* pasa un archivo informando que es el shapefile de zonas a la clase *Principal*.

1.1 La clase *Principal* crea una clase CapaPoligonos correspondiente a las zonas del sistema y se la pasa a la clase *VisualizadorImpl*. Este último recuerda la CapaPoligonos de zonas, a partir de esta:

- crea una CapaPuntos correspondiente a los centroides de cada una de las zonas y lo recuerda.
- con las paradas recordadas, los centroides creados y las zonas, crea una CapaLineas con líneas correspondientes a todas las caminatas que se pueden realizar desde el centroide de una zona a todas las paradas que estén dentro de la zona, y recuerda la CapaLinea correspondiente a las caminatas.

1.1.1 La clase *VisualizadorImpl* envía a SwtFrameSTP una Layer (capa en GeoTools) de la CapaPoligonos de las zonas. SwtFrame ingresa la capa al mapa.

1.1.2 La clase *VisualizadorImpl* envía al *ISimulador* los identificadores de los centroides creados.

1.1.2.1 El *SimuladorImpl* envía los identificadores que llegaron al *ISimulador* a la estructura. La *EstructuraImpl*, crea los centroides en la estructura.

1.1.3 La clase *VisualizadorImpl* envía a SwtFrameSTP una Layer (capa en GeoTools) de la CapaPuntos de los centroides. SwtFrame ingresa la capa al mapa.

1.1.4 La clase *VisualizadorImpl* envía al *ISimulador* un conjunto de caminatas, las cuales están conformadas por <IdCentroide, IdParada,distancia de centoide a parada>.

1.1.4.1 El *SimuladorImpl* envía las 3-tuplas que llegaron al *ISimulador* a la estructura. La *EstructuraImpl*, crea las caminatas y genera la relación entre centroide, parada y caminata en la estructura.

1.1.5 La clase *VisualizadorImpl* envía a SwtFrameSTP una Layer (capa en GeoTools) de la CapaLineas de las caminatas. SwtFrame ingresa la capa al mapa.

5.4.4 - Shapefile de líneas de ómnibus

Una vez que se tienen las zonas, los centroides y las paradas cargados en el sistema, se pueden cargar las líneas con sus recorridos. El archivo de líneas de ómnibus debe contener los siguientes campos: ID (identificador numérico único de la capa de línea), ID_LINEA (nombre de la línea de ómnibus), TIPO (corresponde a si el recorrido de la línea de ómnibus es ida, vuelta o circular) y PARADAS (identificadores ordenados de las paradas del recorrido de la línea de ómnibus, cada parada debe estar separada por ";"). Este último campo no es obligatorio si en la ruta donde se encuentra un archivo de nombre "*Recorridos.csv*" descripto en *4.2 - Transformación de datos*.

Implementación de la carga de líneas de ómnibus, recorridos y tramos

En la Figura 29 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

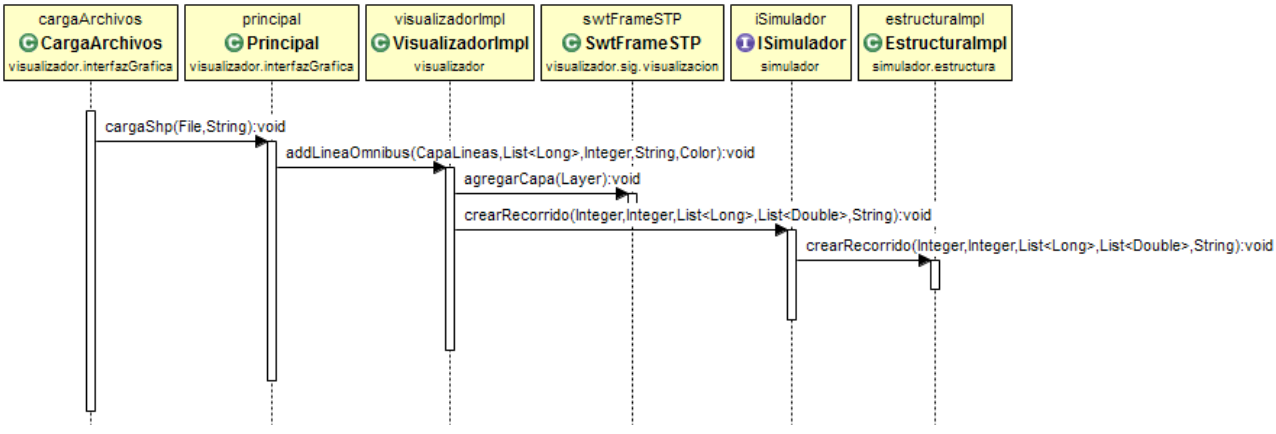


Figura 29 - Diagrama de secuencia de carga líneas, recorridos y tramos

1 La clase *CargaArchivo* pasa un archivo informando que es el shapefile de líneas a la clase *Principal*.
1.1 La clase Principal procesa el shapefile y por cada línea realiza lo siguiente:

- Si existe el archivo "*Recorridos.csv*" obtiene las paradas de la línea a partir de este, si no las obtiene del campo "*Paradas*" del shapefile, como se menciona anteriormente.
- Crea un color de forma random para su representación.
- Crea la *CapaLinea* con el ID y el ID_LINEA, el TIPO (que vienen en el shp), el color creado y una única línea.
    - Envía a la clase *VisualizadorImpl* la *CapaLinea* creada, las paradas en orden que realiza la línea, y el color creado.

Para cada CapaLinea recibida:
1.1.1 La clase *VisualizadorImpl* envía a *SwtFrameSTP* una Layer (capa en GeoTools) de la *CapaLineas* de la línea. *SwtFrameSTP* ingresa la capa al mapa.
1.1.2 La clase *VisualizadorImpl* envía a *ISimulador* el identificador de la línea a crear, el identificador del recorrido a crear, una lista ordenada con identificadores de las paradas por las que pasa el recorrido (las paradas ya fueron cargadas), una lista ordenada de distancias que corresponden a las distancias entre paradas, y el tipo de recorrido que es (IDA, VUELTA o CIRCULAR).
1.1.2.1 El *SimuladorImpl* le envía a la estructura los datos recibidos en el punto anterior. *EstructuraImpl* crea el recorrido; si la línea ya existe asigna el recorrido a la línea según el tipo, si no existe crea una línea y le asigna el recorrido; asocia el recorrido con las paradas y las distancias entre estas para el recorrido.

5.4.5 - Archivo de texto de matriz OD

Contiene las tasas de creación de pasajeros por par centroide origen-destino.

Cada elemento de la matriz OD representa la tasa (la unidad es 1/minutos) para la creación de pasajeros desde un centroide origen O a un centroide destino D, es decir, se crea un pasajero cada 1/matriz[O][D] minutos en promedio para ir de O a D. Las filas de la matriz representan el origen y las columnas el destino.

Implementación de la carga de la matriz OD

En la Figura 30 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

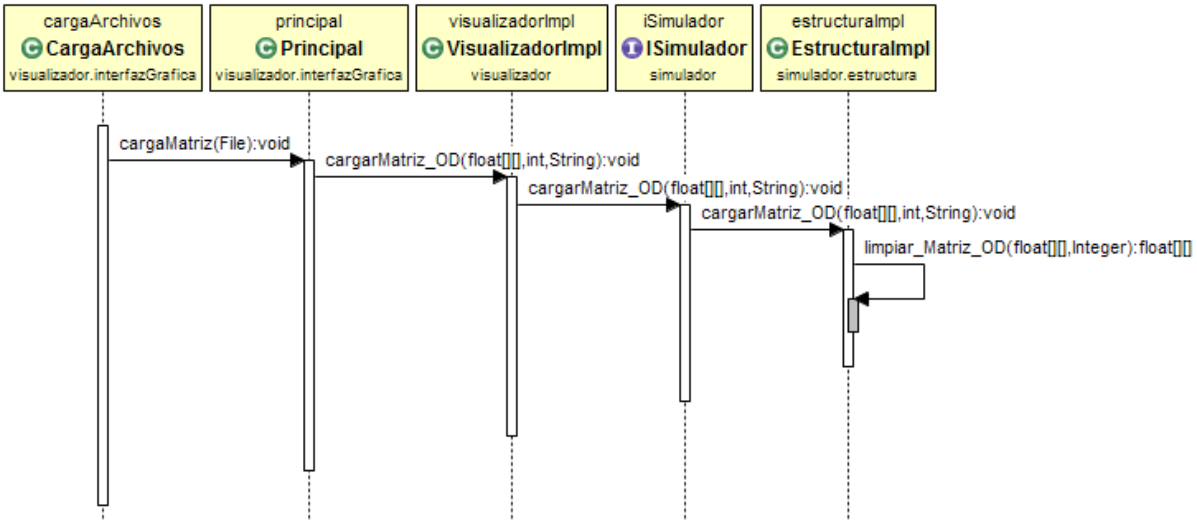


Figura 30 - Diagrama de secuencia de carga matriz OD

1 La clase *CargaArchivo* pasa el archivo correspondiente a la matriz OD a la clase Principal.
1.1 La clase *Principal* crea una matriz donde las filas corresponden a los orígenes y las columnas a los destinos y en cada celda se carga el valor que viene en el archivo y lo envía a la clase *VisualizadorImpl*.
1.1.1 La clase *VisualizadorImpl* envía a *ISimulador* la matriz obtenida.
1.1.1.1 La clase *SimuladorImpl* envía la matriz a la estructura.
1.1.1.1.1 La clase *EstructuraImpl* corrobora para cada elemento distinto de cero de la matriz obtenida, que exista un recorrido que pase por alguna parada de zona origen y tenga una parada posterior en la zona destino. En caso de que no exista modifica el valor de la matriz a cero.[4]

5.4.6 - Archivo de texto de horarios de ómnibus.

El último archivo necesario para cargar el sistema, es el que contiene los horarios en que los ómnibus parten de la parada inicial. Hay dos formas para cargar el horario:

- Frecuencia de salida: Indica cada cuantos minutos sale un ómnibus de determinada línea.
- Tabla de horarios: Indica todos los horarios de salida de los ómnibus en el día.

El formato del archivo es el siguiente:

ID TIPO [frecuencia | horarios]

donde ID corresponde al nombre de la línea (ID_LINEA de la línea de ómnibus), TIPO es el tipo de recorrido (IDA, VUELTA, CIRCULAR), frecuencia es el tiempo en minutos que indica cada cuanto sale un ómnibus y horarios es cada uno de los instantes de tiempo que sale cada ómnibus para la línea separada por espacios con el formato HH:mm.

Implementación de la carga de horarios

En la Figura 31 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

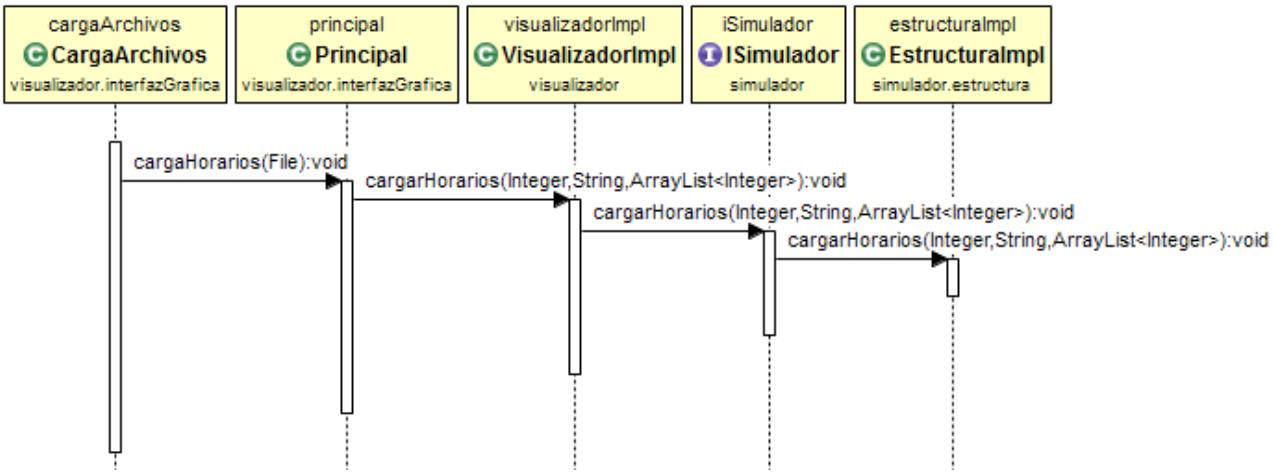


Figura 31 - Diagrama de secuencia de carga de horarios

1 La clase *CargaArchivo* pasa el archivo correspondiente de horarios de ómnibus a la clase Principal.
1.1 La clase *Principal* procesa el archivo y por cada línea de texto del archivo:

- Obtener el Id de la Línea de Ómnibus.
- Obtener el tipo de recorrido de la Línea de Ómnibus.
- Si los horarios vienen en formato de frecuencia, crea una lista con valores múltiplos de la fre-

4 La función limpiar_Matriz_OD elimina la creación de pasajeros para los pares OD, en los cuales no hay un recorrido directo entre las zonas correspondientes. Esto es debido a que para el alcance del proyecto solo se tienen encuenta en el análisis las estretegias implementadas. Para que esto no suceda solo se debe eliminar la condición de borrado en la función.

cuencia que van de 0 a 24*60.

- Si no, crea una lista con los valores de la tabla de horarios pasados a minutos.
- Envía a la clase *VisualizadorImpl* el Id, el tipo y la lista de horarios obtenidos

Para cada horario recibido por *VisualizadorImpl* recibida

1.1.1 La clase *VisualizadorImpl* envía a *ISimulador* los datos obtenidos.

1.1.1.1 La clase *SimuladorImpl* envía los datos obtenidos a la estructura. *EstructuraImpl* carga los horarios al recorrido de la línea Id para el recorrido del tipo obtenido.

5.4.7 - Seleccionar archivos

Hay dos maneras de cargar los archivos generados.
La primera consiste en seleccionar una carpeta donde se encuentran los archivos:

- configuracion.txt (archivo configuración inicial)
- zonas.shp, zonas.dbf y zonas.shx (shapefiles de zonas)
- paradas.shp, paradas.dbf, paradas.shx (shapefile de paradas)
- lineas.shp, lineas.dbf, lineas.shx (sheapefile de líneas de ómnibus)
- matriz.txt (archivo de texto de matriz OD)
- horarios.txt (archivo de texto de horarios de ómnibus)

La segunda forma es seleccionando uno a uno los archivos, sin tener la obligación de tener el mismo nombre, sino que deben respetar los requisitos para cada archivo.
En la Figura 32 , se muestra la pantalla de carga de datos correspondiente.

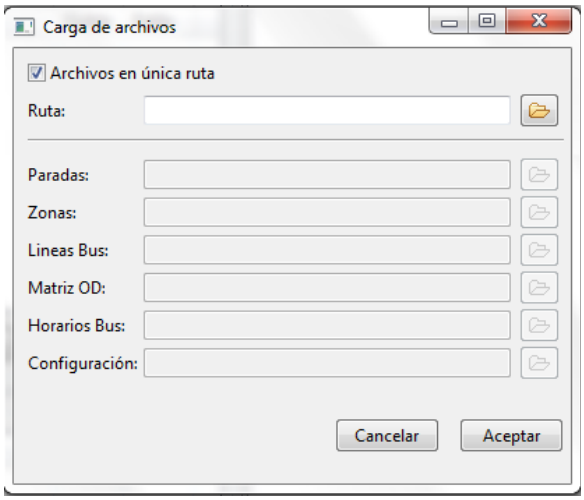


Figura 32 - Carga de archivos.

Luego de cargado los archivos, se da la posibilidad de modificar los porcentajes de estrategias configuradas como se ve en la Figura 33 .

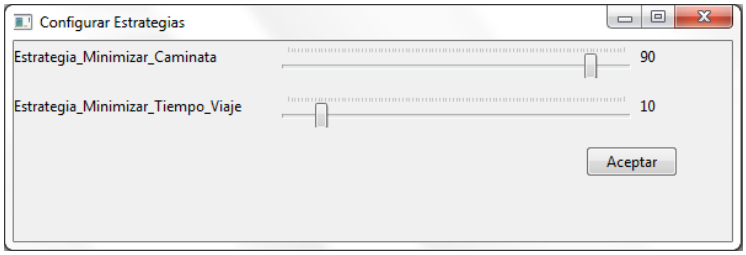


Figura 33 - Configurar Estrategias

Luego de aceptada la configuración ingresada, se procede a la carga descrita. En la Figura 34 se muestra la visualización final del sistema ya iniciado.

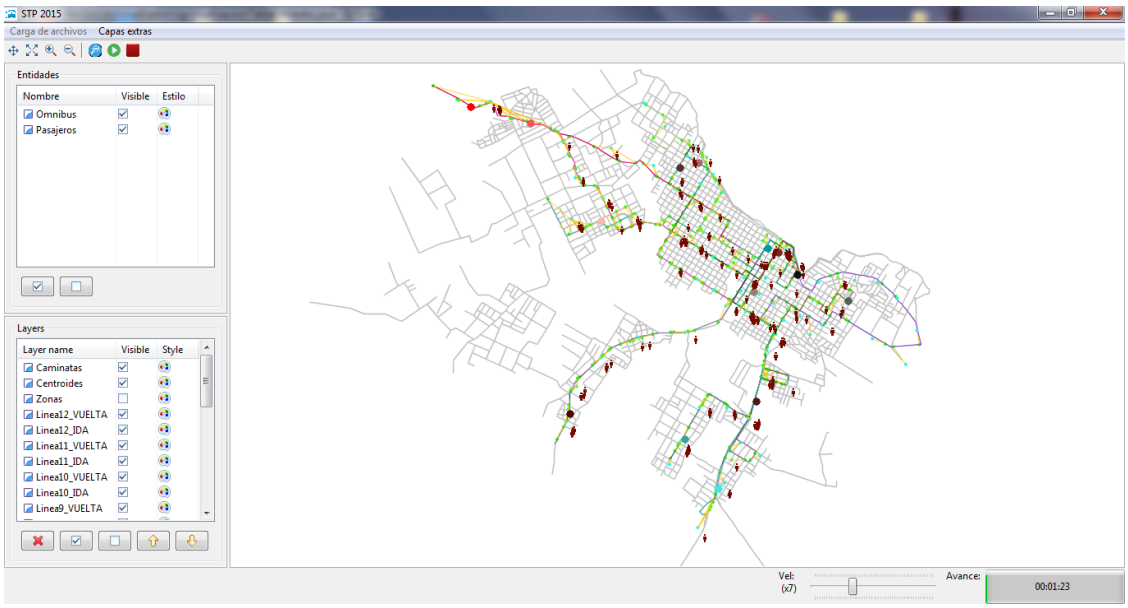


Figura 34 - Visual del sistema una vez cargada e iniciada la simulación.

5.5 - Integración de simulador y visualizador

Hasta el momento se ha visto cómo se comporta el sistema en forma aislada. En esta sección se muestra como se realiza la comunicación entre el simulador y el visualizador para que las ejecuciones estén sincronizadas. A nivel de diseño, los módulos se definieron de forma independiente, y se comunican mediante estructuras de identificadores de elementos. Todo lo que refiere al llenado de dichas estructuras, el cálculo y los componentes de la estrategia del pasajero quedan confinados dentro del módulo de la Estructura. Para ello se utilizan dos hilos de ejecución, uno para el simulador y otro para el visualizador.

En el hilo del simulador, se ejecuta la instancia de DESMO-J, el cual se encarga de realizar el modelo de simulación e informa de los cambios a la interfaz del visualizador. Las acciones posibles a informar al visualizador son: crear pasajeros, situar pasajero en un punto, caminata de pasajero, eliminar pasajero, crear ómnibus, situar ómnibus, mover ómnibus, eliminar ómnibus, mostrar estrategia de pasajero a seguir, finalizar mostrar estrategia de pasajero a seguir y finalizar la ejecución.

El hilo del visualizador se encarga de redibujar el mapa utilizando GeoTools, según la información que le brinda el simulador. También informa al simulador a través de su interfaz sobre las acciones a realizar, las cuales son iniciar, pausar, seguir, cambiar la velocidad y parar la simulación, e informar a cuál pasajero hay que hacerle seguimiento.

Se detallarán los casos de uso utilizados para la interacción entre los componentes, así como la implementación de cada uno de ellos. En el caso del seguimiento de pasajeros, al ser una funcionalidad especial, se describe por separado en la sección *5.6*.

En los diagramas de secuencia se puede ver cuál es la interfaz utilizada para la comunicación entre los componentes. Cuando desde el visualizador se informa al simulador de una acción, se utiliza la interfaz Isimulador; mientras que en el caso de que el simulador informe al visualizador lo hace a través de la interfaz IVisualizador, como se muestra en la Figura 25 - *Interacción entre componentes*.

A continuación se muestran los casos de usos cuando se ejecuta el sistema con cantidad de replicaciones

igual a uno. Esto significa que se debe visualizar la simulación. En caso de que cantidad de replicaciones es mayor a uno, no se visualiza la simulación, la ejecución se realiza a la máxima velocidad posible y en el visualizador solo se informa en el número de replicación en la que se está.

5.5.1 - Caso de uso: Iniciar Simulación

Descripción:

Inicia la ejecución del sistema.

Actores:

usuario, visualizador y simulador.

Precondiciones:

El sistema debe estar cargado y no iniciado.

Flujo Normal:

1. El caso de uso comienza cuando el usuario selecciona la opción iniciar la simulación.
2. El visualizador informa al simulador que se inicia la simulación, y empieza a ejecutar el hilo de dibujar el mapa.
3. El simulador empieza con la simulación.

Flujo Alternativo:

No hay

Poscondiciones:

El sistema está iniciado y ejecutando.

Implementación

En la Figura 35 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

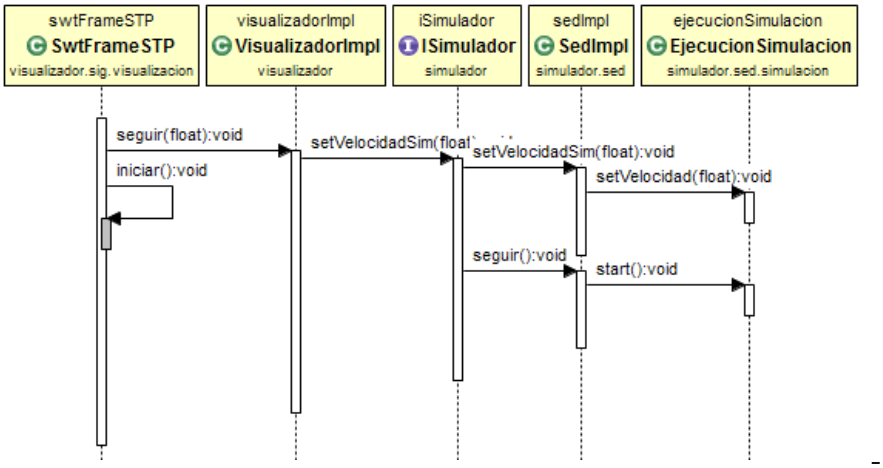


Figura 35 - Diagrama de secuencia: Iniciar Simulación

1 Cuando el *SwtFrameSTP* detecta que se solicita iniciar la simulación, le informa a *VisualizadorImpl* que debe seguir la simulación con determinada velocidad.

1.1 El *VisualizadorImpl* informa a *ISimulador* cual es la velocidad con la que debe continuar la simulación.

1.1.1 El *SimuladorImpl* informa al *SedImpl* con cual es la velocidad a simular.

1.1.1.1 El *SedImpl* informa a *EjecucionSimulación* con cual es la velocidad a simular. Este último recuerda el valor informado.

1.1.2 El *SimuladorImpl* informa que debe seguir con la simulación.

1.1.2.1 El *SedImpl* detecta que no se inició la simulación e informa al *EjecucionSimulacion* que empiece a ejecutar el hilo del simulador.

2 El *SwtFrameSTP* ejecuta el hilo de redibujar.

5.5.2 - Caso de uso: Pausar Simulación

Descripción:

Se pausa la simulación

Actores:

usuario, visualizador y simulador.

Precondiciones:

El sistema debe estar iniciado y ejecutando.

Flujo Normal:

1. El caso de uso comienza cuando el usuario selecciona la opción pausar la simulación.
2. El visualizador informa al simulador que se pausa la simulación y deja de dibujar el mapa.
3. El simulador pausa la simulación.

Flujo Alternativo:

No hay

Poscondiciones:

El sistema está iniciado y pausado.

Implementación

En la Figura 36 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

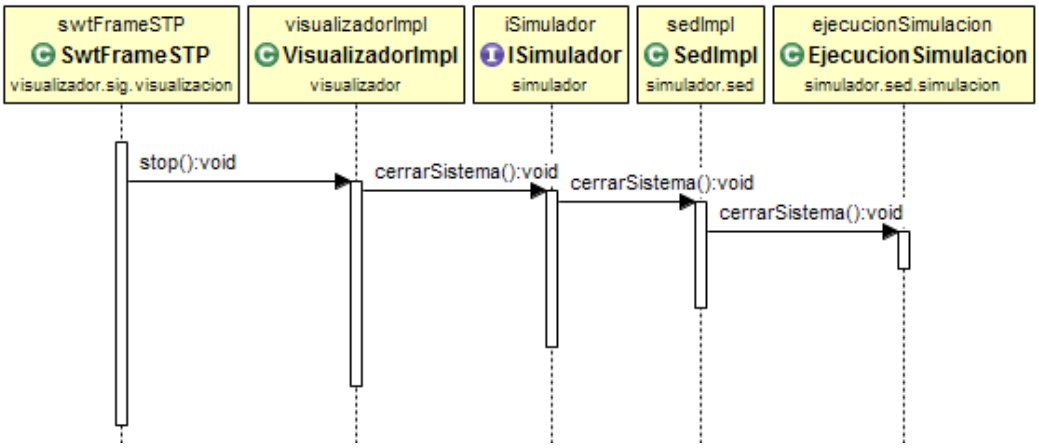


Figura 36 - Diagrama de secuencia: Pausar Simulación

1 Cuando el *SwtFrameSTP* detecta que se solicita pausar la simulación, le informa a *VisualizadorImpl* que se pausa la simulación.

1.1 El *VisualizadorImpl* deja de re-dibujar el mapa y le informa al *ISimulador* que se pausó la simulación.

1.1.1 El *SimuladorImpl* informa al *SedImpl* que se pausa la simulación.

1.1.1.1 El *SedImpl* informa a *EjecucionSimulación* que debe pausar la simulación. Este último, pausa la simulación asignando la velocidad igual a cero.

5.5.3 - Caso de uso: Seguir Simulación

Descripción:

Se pausa la simulación

Actores:

usuario, visualizador y simulador.

Precondiciones:

El sistema debe estar iniciado y pausado.

Flujo Normal:

1. El caso de uso comienza cuando el usuario selecciona la opción seguir la simulación.
2. El visualizador informa al simulador que sigue la simulación, y nuevamente dibuja el mapa.
3. El simulador continúa con la simulación.

Flujo Alternativo:

1.A)

1. El usuario modifica la velocidad de la simulación.
2. Vuelve al paso 1 del Flujo Normal.

Poscondiciones:

El sistema está iniciado y ejecutando.

Implementación

En la Figura 37 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

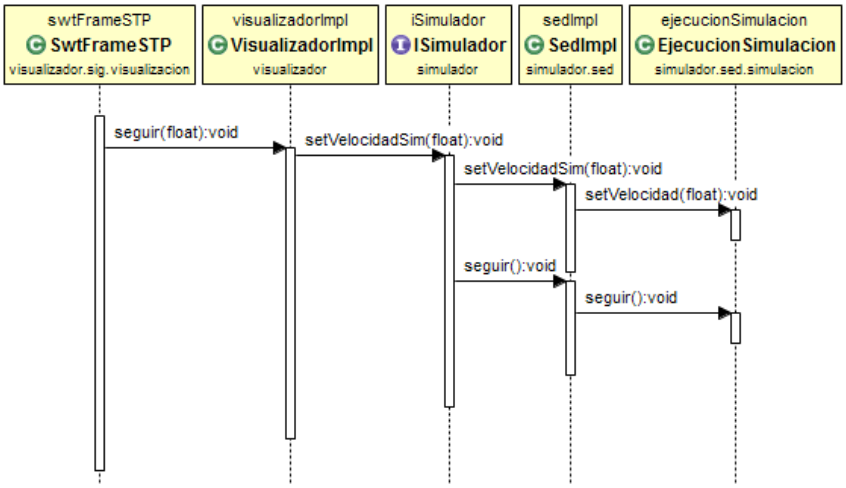


Figura 37 - Diagrama de secuencia: Seguir Simulación

1 Cuando el *SwtFrameSTP* detecta que se solicita continuar con la simulación, le informa a *VisualizadorImpl* que debe seguir la simulación con determinada velocidad.

1.1 El *VisualizadorImpl* le informa al *ISimulador* cual es la velocidad con la que debe continuar la simulación.

1.1.1 El *SimuladorImpl* informa al *SedImpl* con cual es la velocidad a simular.

1.1.1.1 El *SedImpl* informa a *EjecucionSimulación* con cual es la velocidad a simular. Este último recuerda el valor informado

1.1.2 El *SimuladorImpl* informa que debe seguir con la simulación.

1.1.2.1 El *SedImpl* informa al *EjecucionSimulacion* que continúe con la simulación. Este último, continúa con la simulación asignando la velocidad el valor recordado.

5.5.4 - Caso de uso: Parar simulación

Descripción:

Para la simulación

Actores:

usuario, visualizador y simulador.

Precondiciones:

El sistema debe estar iniciado.

Flujo Normal:

1. El caso de uso comienza cuando el usuario selecciona la opción parar la simulación.
2. El visualizador informa al simulador que se para la simulación, y deja de dibujar el mapa.
3. El simulador finaliza la simulación.

Flujo Alternativo:

No hay

Poscondiciones:

El sistema está finalizado.

Implementación

En la Figura 38 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

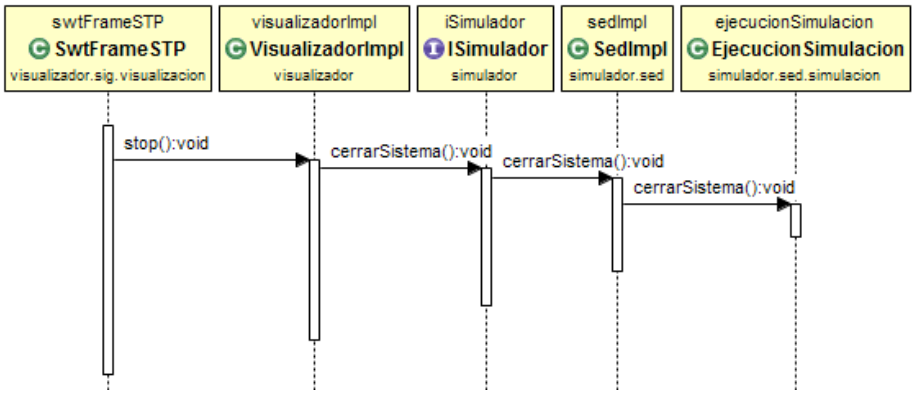


Figura 38 - Diagrama de secuencia: Parar Simulación

1 Cuando el *SwtFrameSTP* detecta que se solicita parar la simulación, le informa a *VisualizadorImpl* que se debe parar la simulación.

1.1 El *VisualizadorImpl* deja de re-dibujar el mapa y le informa al *ISimulador* que termine la simulación.

1.1.1.1 El *SimuladorImpl* informa al *SedImpl* que debe terminar la simulación.

1.1.1.2 El *SedImpl* informa a *EjecucionSimulación* que debe terminar la simulación. Este último, termina la simulación y genera los reportes de la corrida realizada.

5.5.5 - Caso de uso: Crear pasajero

Descripción:

Se crea un pasajero nuevo en el sistema

Actores:

visualizador y simulador.

Precondiciones:

El sistema debe estar iniciado y ejecutando.

Flujo Normal:

1. El caso de uso comienza cuando el simulador crea un pasajero nuevo.
2. El simulador informa al visualizador que se creó un pasajero en un centroide “C”.
3. El visualizador ingresa al pasajero creado en el centroide “C” para su representación en el mapa .

Flujo Alternativo:

No hay

Poscondiciones:

No hay

Implementación

En la Figura 39 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

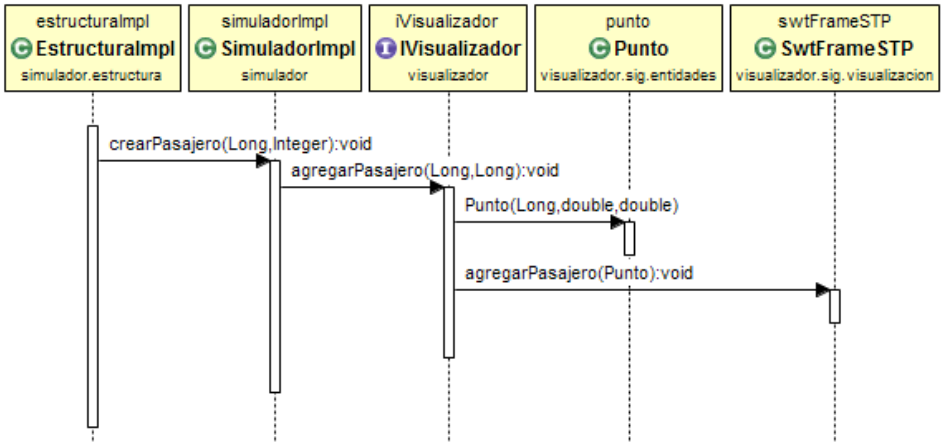


Figura 39 - Diagrama de secuencia: Crear Pasajero

1 Cuando desde el simulador se crea un pasajero, la *EstructuraImpl* le informa al *SimuladorImpl* que se creó un pasajero, pasando como parámetros el identificador con el cual se creó y el identificador del centroide origen de este.

1.1 El *SimuladorImpl* le informa al *IVisualizador* que debe crear un pasajero con determinado identificador y en el centroide indicado.

1.1.1 El *Visualizador* obtiene las coordenadas del centroide indicado y crea un *Punto* con el identificador del pasajero y las coordenadas obtenidas.

1.1.2 El *Visualizador* le informa al *SwtFrameSTP* que agregue al punto como un pasajero. El *SwtFrameSTP* agrega al punto como pasajero del sistema.

5.5.6 - Caso de uso: Caminata pasajero

Descripción:

Un pasajero realiza una caminata desde donde se encuentra a una parada o un centroide.

Actores:

visualizador y simulador.

Precondiciones:

El sistema debe estar iniciado y ejecutando, existe un pasajero “P”, y una parada “p” o un centroide “c”.

Flujo Normal:

1. El caso de uso comienza cuando el simulador indica que un pasajero realiza una caminata desde donde se encuentra a una parada “p” o a un centroide “c”.
2. El simulador informa al visualizador que el pasajero “P” va a realizar una caminata a la parada “p” o al centroide “c” y la velocidad ”v” en que lo realiza.
3. El visualizador genera el recorrido desde el lugar donde se encuentra “P” al destino informado, y realiza ese recorrido a la velocidad ”v”.

Flujo Alternativo:

No hay

Poscondiciones:

No hay

Implementación

En la Figura 40 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

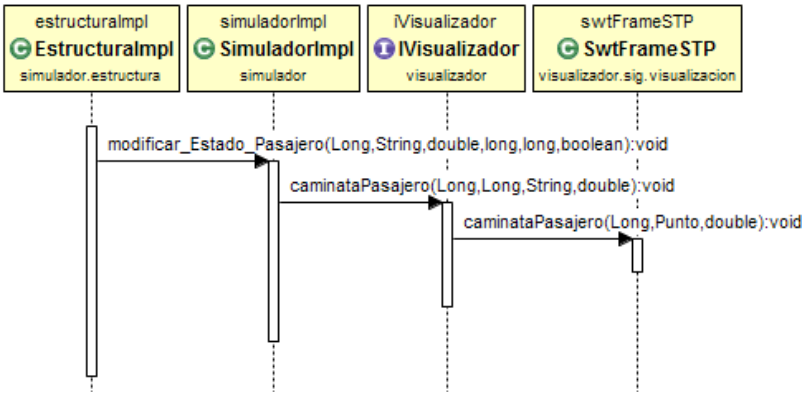


Figura 40 - Diagrama de secuencia: Caminata Pasajero

1 Cuando desde el simulador, un pasajero tiene que realizar una caminata; la *EstructuraImpl* le informa al *SimuladorImpl* que se modifica el estado del pasajero con los siguientes parámetros:

- identificador del pasajero que empieza la caminata
- el estado actual del pasajero. En este caso “Caminando”.
- la velocidad con la que realiza la caminata.
- el identificador de la parada en caso de que tenga que caminar hasta una parada.
- el identificador del centroide en caso de que tenga que caminar hasta el centroide final.
- booleano que indica si la acción corresponde a una parada o un centroide.

1.1 El *SimuladorImpl* le informa al *IVisualizador* que un pasajero con determinado identificador realiza una caminata a un centroide o parada con determinada velocidad.

1.1.1 El *Visualizador* obtiene el *Punto* correspondiente a la parada o al centroide indicado e informa al *SwtFrameSTP* que el pasajero debe realizar una caminata a ese *Punto* con determinada velocidad.

5.5.7 - Caso de uso: Situar pasajero

Descripción:

Se sitúa a un pasajero en una parada o un centroide.

Actores:

visualizador y simulador.

Precondiciones:

El sistema debe estar iniciado y ejecutando. Existe un pasajero “P”, y una parada “p” o un centroide “c”

Flujo Normal:

1. El caso de uso comienza cuando el simulador detecta que un pasajero “P” llega a un parada “p” o a un centroide “c”.
2. El simulador informa al visualizador que el pasajero “P” llegó al centroide “c” o a la parada “p”.
3. El visualizador dibuja en el mapa al pasajero en el centroide “c” o a la parada “p”, según lo indicado.

Flujo Alternativo:

No hay

Poscondiciones:

No hay

Implementación

En la Figura 41 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

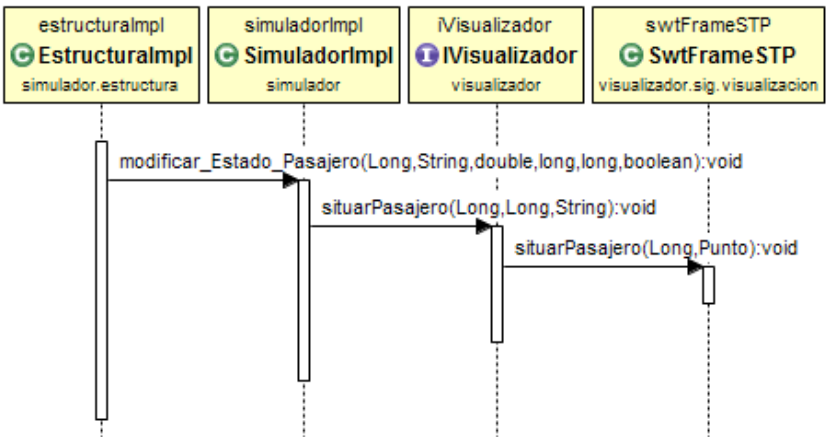


Figura 41 - Diagrama de secuencia: Situar Pasajero

1 Cuando desde el simulador, un pasajero llega a una parada o un centroide; la *EstructuraImpl* le informa al *SimuladorImpl* que se modifica el estado del pasajero con los siguientes parámetros:

- identificador del pasajero
- el estado actual del pasajero. En este caso “Esperando”.
- la velocidad del pasajero. En este caso no se utiliza.
- el identificador de la parada en caso de que llegue a una parada.
- el identificador del centroide en caso de que llegue a un centroide.
- booleano que indica si la acción corresponde a una parada o un centroide

1.1 El *SimuladorImpl* le informa al *IVisualizador* que un pasajero con determinado identificador llega a un centroide o una parada.

1.1.1 El Visualizador obtiene el *Punto* correspondiente a la parada o al centroide indicado e informa al *SwtFrameSTP* que el pasajero llega al *Punto*. El *SwtFrameSTP* asigna las coordenadas del pasajero de forma aleatoria, en un entorno determinado al *Punto* de llegada.

5.5.8 - Caso de uso: Eliminar pasajero

Descripción: Elimina un pasajero del sistema
Actores: visualizador y simulador.
Precondiciones: El sistema debe estar iniciado y ejecutando. Existe un pasajero “P”.
Flujo Normal:

1. El caso de uso comienza cuando el simulador detecta que un pasajero “P” llega al final de su viaje.
2. El simulador elimina al pasajero e informa al visualizador que debe eliminar el pasajero “P”.
3. El visualizador elimina al pasajero “p”.

Flujo Alternativo: No hay
Poscondiciones: No hay

Implementación

En la Figura 42 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

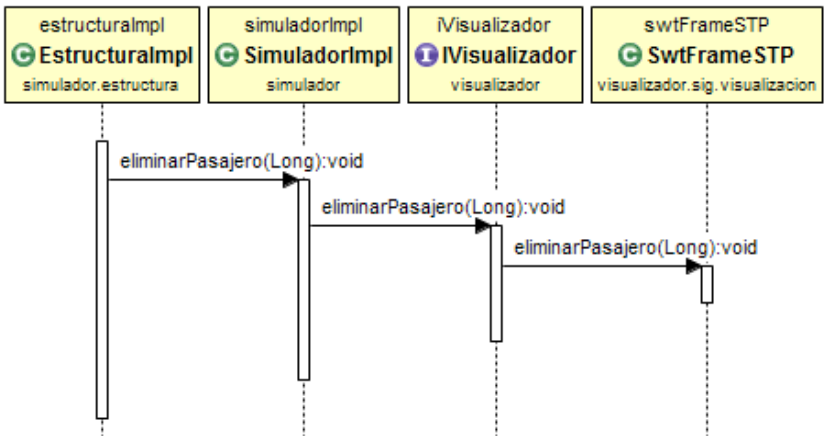


Figura 42 - Diagrama de secuencia: Eliminar Pasajero

1 Cuando desde el simulador, un pasajero llega al centroide final, la *EstructuraImpl* le informa al *SimuladorImpl* que debe eliminar el pasajero con determinado identificador.
1.1 El *SimuladorImpl* le informa al *IVisualizador* que elimine el pasajero indicado.
1.1.1 El *Visualizador* le informa al *SwtFrameSTP* que elimine al pasajero. El *SwtFrameSTP* saca al Punto que corresponde con el identificador del pasajero de la lista de pasajeros.

5.5.9 - Caso de uso: Crear ómnibus

Descripción:

Crea un ómnibus para realizar un recorrido.

Actores:

visualizador y simulador.

Precondiciones:

El sistema debe estar iniciado y ejecutando.

Flujo Normal:

1. El caso de uso comienza cuando el simulador crea un ómnibus “B”.
2. El simulador informa al visualizador que se creó un ómnibus “B” de la línea “l”.
3. El visualizador ingresa al ómnibus “B” para realizar el recorrido de la línea “l”.

Flujo Alternativo:

No hay

Poscondiciones:

No hay

Implementación

En la Figura 43 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

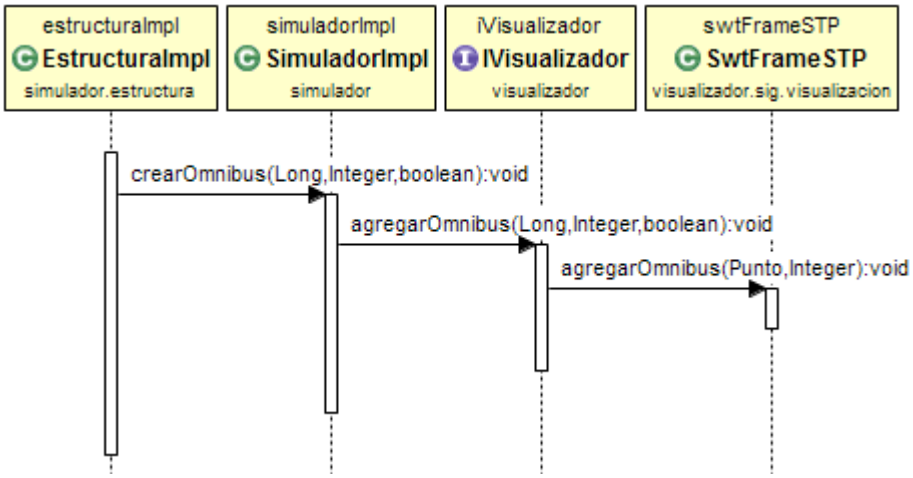


Figura 43 - Diagrama de secuencia: Crear Omnibus

1 Cuando desde el simulador se crea un ómnibus, la *EstructuraImpl* le informa al *SimuladorImpl* que se creó un pasajero, pasando como parámetros el identificador con el cual se creó y el identificador de la línea a la cual pertenece el ómnibus y si realiza un recorrido de vuelta.

1.1 El *SimuladorImpl* le informa al *IVisualizador* que debe crear un ómnibus con determinado identificador, para la línea de ómnibus indicada y si es un recorrido de vuelta.

1.1.1 El *Visualizador* obtiene el recorrido a partir del identificador de la línea y si es un recorrido de vuelta. En caso de que la línea tenga un solo recorrido (recorrido circular), el último parámetro no es tenido en cuenta. Luego obtiene las coordenadas de la parada inicial del recorrido, y crea un *Punto* con el identificador del ómnibus y con las coordenadas obtenidas. El *Visualizador* le informa al *SwtFrameSTP* que agregue al ómnibus en el sistema. El *SwtFrameSTP* agrega al ómnibus a la lista de ómnibus del sistema para dibujar.

5.5.10 - Caso de uso: Situar ómnibus

Descripción:

Se sitúa a un ómnibus en una parada.

Actores:

visualizador y simulador.

Precondiciones:

El sistema debe estar iniciado y ejecutando. Existe un ómnibus “B” y una parada “p”.

Flujo Normal:

1. El caso de uso comienza cuando el simulador detecta que un ómnibus “B” llega a un parada “p”
2. El simulador informa al visualizador que el ómnibus “B” llegó a la parada “p”.
3. El visualizador dibuja en el mapa al pasajero en el la parada “p”.

Flujo Alternativo:

No hay

Poscondiciones:

No hay

Implementación

En la Figura 44 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

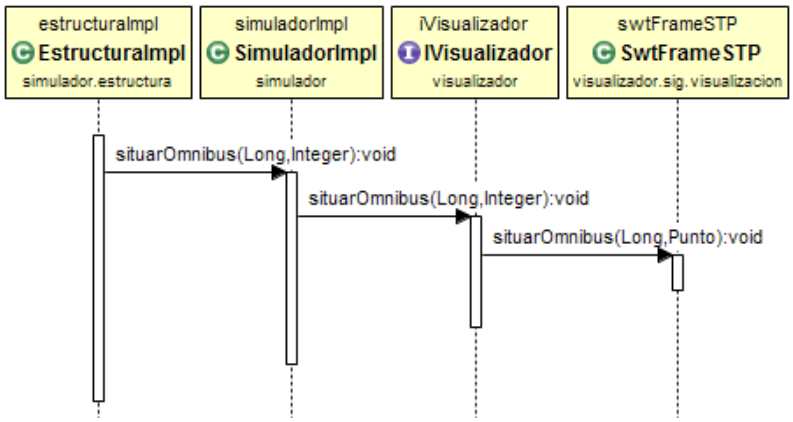


Figura 44 - Diagrama de secuencia: Situar Ómnibus

1 Cuando desde el simulador, un ómnibus llega a una parada, la *EstructuraImpl* le informa al *SimuladorImpl* que debe situar al ómnibus en la parada a través de los identificadores de cada uno.

1.1 El *SimuladorImpl* le informa al *IVisualizador* que un ómnibus con determinado identificador llega a una parada.

1.1.1 El *Visualizador* obtiene el Punto correspondiente a la parada e informa al *SwtFrameSTP* que el ómnibus llega al Punto. El *SwtFrameSTP* setea las coordenadas del punto de la parada a las del ómnibus.

5.5.11 - Caso de uso: Mover ómnibus

Descripción:

Un ómnibus realiza una recorrido de parada a parada.

Actores:

visualizador y simulador.

Precondiciones:

El sistema debe estar iniciado y ejecutando, y existe ómnibus “B” de la línea “l”, una parada “p” y una parada “p1”. Las paradas “p” y “p1” pertenecen a la línea “l” y “p” está antes en el recorrido que “p1”.

Flujo Normal:

1. El caso de uso comienza cuando el *simulador* indica que un ómnibus “B” debe realizar un recorrido de la parada “p” a la parada “p1”.
2. El simulador informa al *visualizador* que el ómnibus “B” de la línea “l” va a realizar un recorrido desde la parada “p” a la parada “p1” y a una velocidad ”v”.
3. El *visualizador* genera el recorrido desde la parada “p” a la parada “p1” de la línea “l” para el ómnibus “B” y realiza ese recorrido a la velocidad ”v”.

Flujo Alternativo:

No hay

Poscondiciones:

No hay

mplementación

En la Figura 45 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

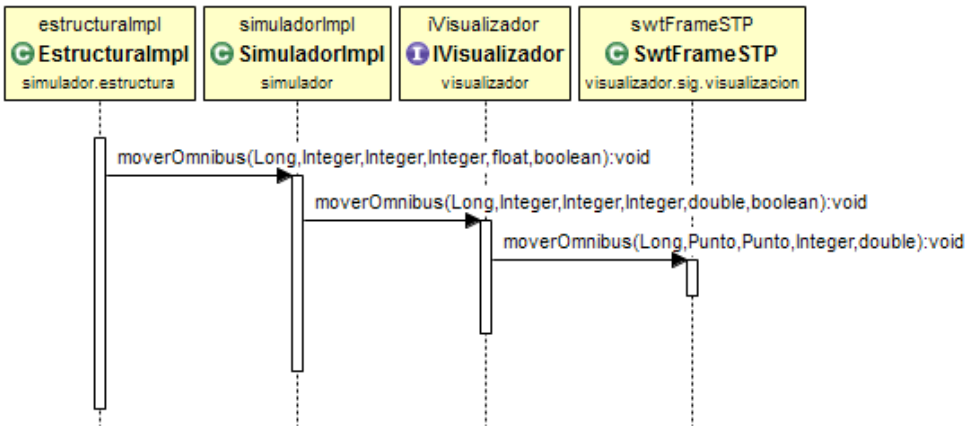


Figura 45 - Diagrama de secuencia: Mover Ómnibus

1 Cuando desde el simulador, un ómnibus sale de una parada hacia otra, la *EstructuraImpl* le informa al *SimuladorImpl* que el ómnibus debe viajar hasta la próxima parada. Para ello utiliza los siguientes parámetros:

- identificador del ómnibus, identificador de la línea del ómnibus, si es recorrido de vuelta
- identificador de parada actual, identificador de próxima parada
- velocidad del ómnibus en el tramo

1.1 El *SimuladorImpl* le informa al *IVisualizador* los parámetros descritos en el paso anterior.

1.1.1 El *Visualizador* obtiene el *Punto* correspondiente a la parada actual y el de la parada siguiente. Luego obtiene a partir del identificador de la línea de ómnibus y del parámetro si es recorrido de vuelta. Por último informa al *SwtFrameSTP* el identificador del ómnibus, los puntos de las paradas actual y siguiente, el recorrido del ómnibus y la velocidad del tramo. El *SwtFrameSTP* crea un sub recorrido desde el punto de la parada inicial al punto de la parada final. Y se lo asigna al ómnibus para que lo realice a la velocidad indicada.

5.5.12 - Caso de uso: Eliminar ómnibus

Descripción:

Elimina un ómnibus del sistema

Actores:

visualizador y simulador.

Precondiciones:

El sistema debe estar iniciado y ejecutando. Existe un ómnibus “B”.

Flujo Normal:

1. El caso de uso comienza cuando el simulador detecta que un ómnibus “B” termina su recorrido.
2. El simulador elimina al ómnibus e informa al visualizador que debe eliminar el ómnibus “B”.
3. El visualizador elimina al ómnibus “B”.

Flujo Alternativo:

No hay

Poscondiciones:

No hay

Implementación

En la Figura 46 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

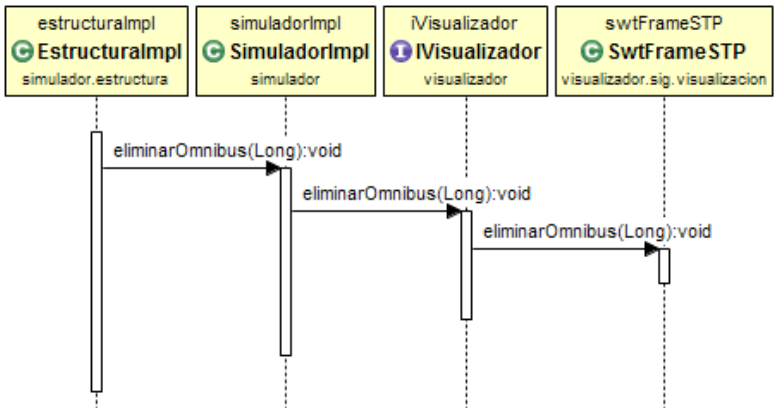


Figura 46 - Diagrama de secuencia: Eliminar Ómnibus

1 Cuando desde el simulador, un ómnibus llega al final de su recorrido, la *EstructuraImpl* le informa al *SimuladorImpl* que debe eliminar el ómnibus con determinado identificador.

1.1 El *SimuladorImpl* le informa al *IVisualizador* que elimine el ómnibus indicado.

1.1.1 El *Visualizador* le informa al *SwtFrameSTP* que elimine el ómnibus. El *SwtFrameSTP* saca al Punto que corresponde con el identificador del ómnibus de la lista de ómnibus.

5.5.13 - Caso de uso: Finalizar simulación

Descripción:

Luego de pasado el tiempo configurado de la simulación, se termina la misma.

Actores:

visualizador y simulador.

Precondiciones:

El sistema debe estar iniciado y ejecutando.

Flujo Normal:

1. El caso de uso comienza cuando el simulador detecta que pasó el tiempo de la simulación.
2. El simulador genera los reportes configurados e informa al visualizador que debe terminar la simulación.
3. El visualizador deja de dibujar el mapa.

Flujo Alternativo:

No hay

Poscondiciones:

El sistema está finalizado.

Implementación

En la Figura 47 se muestra el diagrama de secuencia de la funcionalidad y a continuación está la explicación del diagrama.

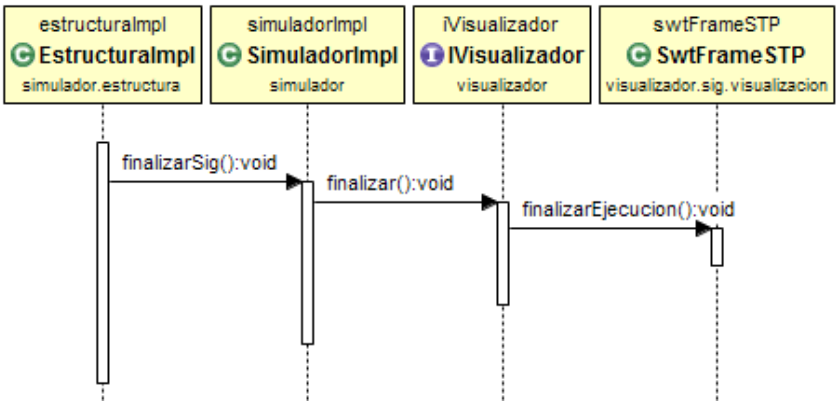


Figura 47 - Diagrama de secuencia: Finalizar Simulación

1 Cuando desde el simulador se termina el tiempo de la ejecución; la *EstructuraImpl* le informa al *SimuladorImpl* que terminó la ejecución.

1.1 El *SimuladorImpl* le informa al *IVisualizador* que terminó la ejecución.

1.1.1 El *Visualizador* le informa al *SwtFrameSTP* que terminó la ejecución El *SwtFrameSTP* deja de redibujar el mapa.

5.6 - Seguimiento de un pasajero

El seguimiento de un pasajero es una funcionalidad de especial importancia para este proyecto. Es por ese motivo que se decidió describirla en una sección independiente, aunque también es una interacción entre el simulador y el visualizador. Es de necesidad poder indicar de forma visual las opciones de viajes que decide realizar un pasajero según su estrategia asignada. Para poder cumplir con tal funcionalidad, es necesario contar con tres etapas diferentes:

1. La selección del pasajero a seguir.
2. La creación del pasajero y por lo tanto la visualización de su estrategia.
3. La eliminación del pasajero y de la visualización de la estrategia.

A continuación se muestran el diseño y la implementación de las tres etapas mencionadas.

5.6.1 - Seleccionar Pasajero a seguir

Descripción:

Seleccionar un pasajero para realizarle seguimiento.

Actores:

usuario, visualizador y simulador.

Precondiciones:

El sistema debe estar pausado o no iniciado.

Flujo Normal:

1. El caso de uso comienza cuando el usuario selecciona la opción de seguimiento de pasajero.
2. El visualizador muestra todas las zonas.
3. El usuario selecciona una zona “O” como origen.
4. El visualizador informa al simulador que se seleccionó la zona “O” como origen.
5. El simulador devuelve todas zonas destino que se pueden llegar desde “O”.
6. El visualizador muestra las zonas destinos.
7. El usuario selecciona una zona destino “D” y un número “n” que significa el n-ésimo pasajero a realizar seguimiento.
8. El visualizador informa al simulador que se seleccionó como zona destino a “D” y el número “n”.

Flujo Alternativo:

G1-

1. El usuario cancela la solicitud
2. El simulador olvida todos los datos informados.

7A-

1. El usuario selecciona otra zona origen ”O”.
2. Vuelve a 4 del Flujo Normal.

Poscondiciones:

El simulador recuerda el número “n” para realizar el seguimiento al n-ésimo pasajero con origen “O” y destino “D” a partir de ese momento. El sistema sigue en el estado en que estaba.

Descripción de implementación

En la primera etapa (con el sistema aún no INICIADO o en estado de PAUSA) se dan opciones para seleccionar un centroide origen a través de sus identificadores. No permite la selección de cualquiera de estos, sino sólo aquellos en donde la matriz OD tenga algún elemento de una fila (orígenes) distinto de cero. Esto significa que desde ese centroide en algún momento se va a crear algún pasajero.

A continuación se habilita la selección e ingreso de un centroide destino. Esta vez tomando en cuenta aquellos que configuran un destino válido a partir del origen seleccionado anteriormente. Esto significa que en la fila de la matriz OD correspondiente al centroide seleccionado, muestra todos los identificadores de las

columnas que no tengan el valor cero.

Para completar la acción se habilita el ingreso del número de pasajero al que se desea hacer el seguimiento. En la Figura 48 se muestra la pantalla para ingresar los datos mencionados.

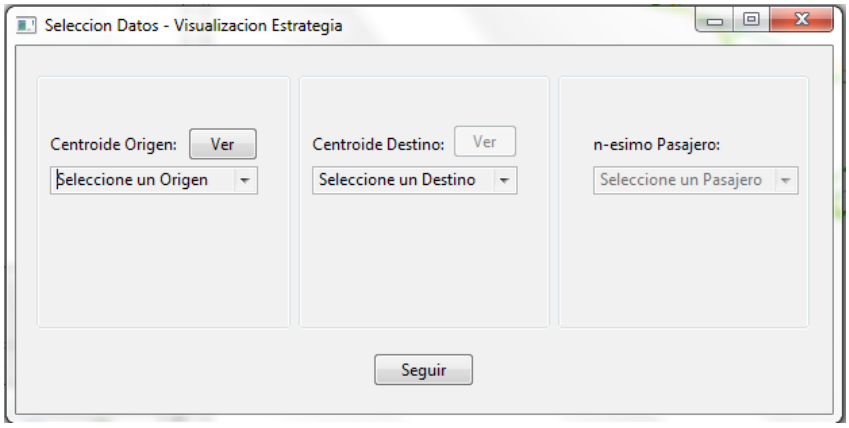


Figura 48 - Selección de pasajero

Una vez completada la captura de datos a nivel de Visualizador se genera una nueva CapaPoligonos: “ZonasTemp” que tiene 2 elementos, la zonas origen y destino seleccionadas (Ver Figura 49). Esta capa queda cargada en MapContent.

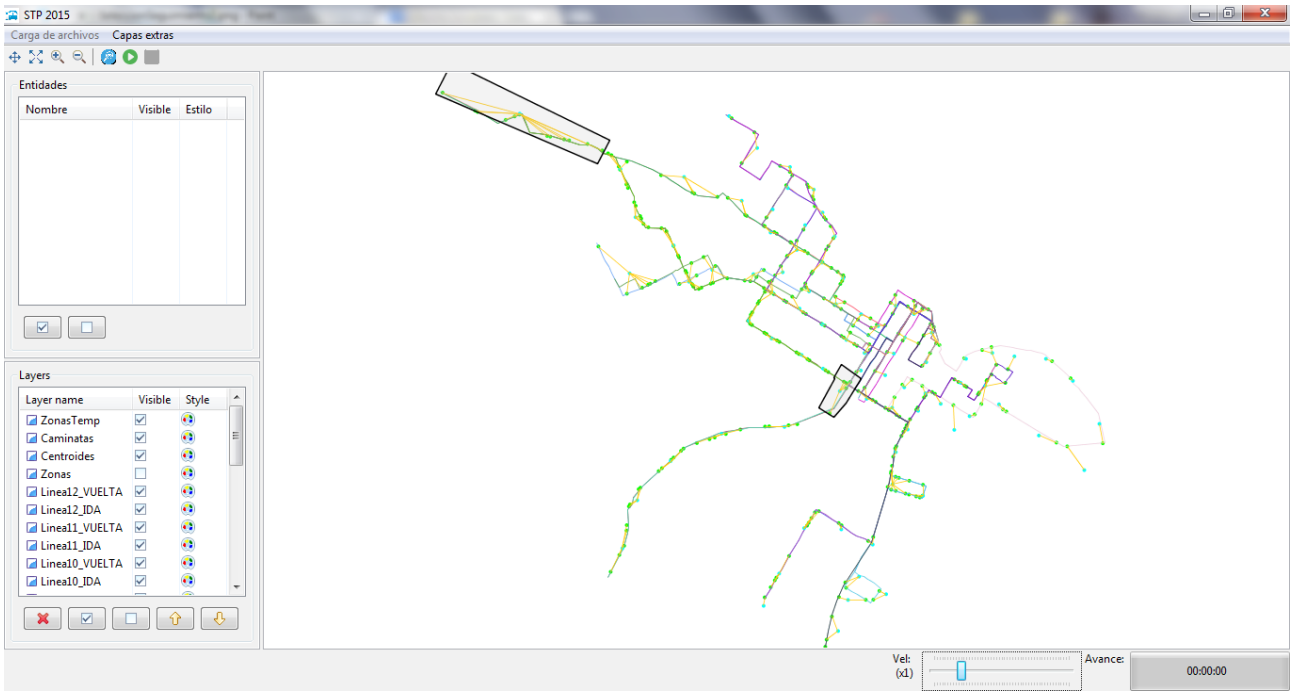


Figura 49 - Mapa luego de la selección

A nivel de estructura del simulador, en MPasajeros guarda el valor de los centroides origen y destino seleccionados y número N de pasajero a seguir a partir de ese momento para ese par de centroides. Y cada vez que se crea un pasajero para este par de centroides, se corrobora si es el n-ésimo pasajero creado luego de configurada la selección.

5.6.2 - Mostrar estrategia de pasajero a seguir

Descripción:

Muestra la estrategia del pasajero seleccionado para su seguimiento.

Actores:

visualizador y simulador.

Precondiciones:

El sistema debe estar iniciado y ejecutando. Existe un pasajero “P” que fue seleccionado a realizarle seguimiento.

Flujo Normal:

1. El caso de uso comienza cuando el simulador detecta la creación del pasajero “P” al cual se le quiere hacer seguimiento.
2. El simulador informa al visualizador los caminos que puede realizar el pasajero “P”, según su estrategia.
3. El visualizador dibuja los caminos informados.

Flujo Alternativo:

No hay

Poscondiciones:

No hay

Descripción de implementación

Cuando el manejador de pasajeros (MPasajeros) detecta que se crea el pasajero a seguir, le solicita a este la información necesaria para realizar la representación de su plan de viaje según su estrategia. La estrategia brinda este servicio a través de la operación:

getParametrosDibujoEstrategia(idCentroide, List<[idCentroide,idParada]>, List<[idParada, idParada]>, List<[idLinea,tipoRecorrido, idParada, idParada, bandera]>)

Donde:

idCentroide: es el centroide origen del pasajero. Este es un parámetro de entrada.

List<[idCentroide, idParada]> lis_centroide_parada: lista de pares de identificador de Centroide e identificador de Parada. Este es un parámetro de salida.

List<[idParada, idParada]> lis_parada_parada: lista de pares de identificadores de paradas. Este es un parámetro de salida.

List<[idLinea,tipoRecorrido, idParada, idParada, bandera]> lis_tomo_omnibus: lista de 5-tupla de identificador de línea, tipo de línea, identificador de parada inicial, identificador de parada final, booleano si la acción anterior es tomar un ómnibus. Este es un parámetro de salida.

El algoritmo realiza una recorrida DFS[5] a través del árbol -visto en la subsección *5.2.2*- que genera una estrategia. Se toman en cuenta tres acciones (IAccion) como determinantes dentro de una estrategia: “CaminarACentroide”, “TomarOmnibus” y “CaminarAParada”, ya que son las únicas que implican un desplazamiento del pasajero en su viaje.

Cada vez que se procesa un Nodo de la estrategia realiza lo siguiente:

- Busca todas las caminatas desde el centroide origen a una parada y de una parada al centroide destino y los ingresa en lis_centroide_parada.
- Busca todas las caminatas de paradas a paradas y las ingresa en lis_parada_parada.
- Busca todas las líneas a tomar e ingresa en lis_tomo_omnibus para cada una, el identificador de la línea, el tipo del recorrido, la parada en que toma el ómnibus, la parada en que se baja del ómnibus y si la acción anterior fue tomar un ómnibus.

5 DFS (Depth First Search) es un algoritmo que permite recorrer todos los nodos de un grafo o árbol de manera ordenada, pero no uniforme. Dado un nodo de partida, recorre el grafo en profundidad [13].

Una vez finalizada la operación, los parámetros de salida cargados se envían al visualizador.

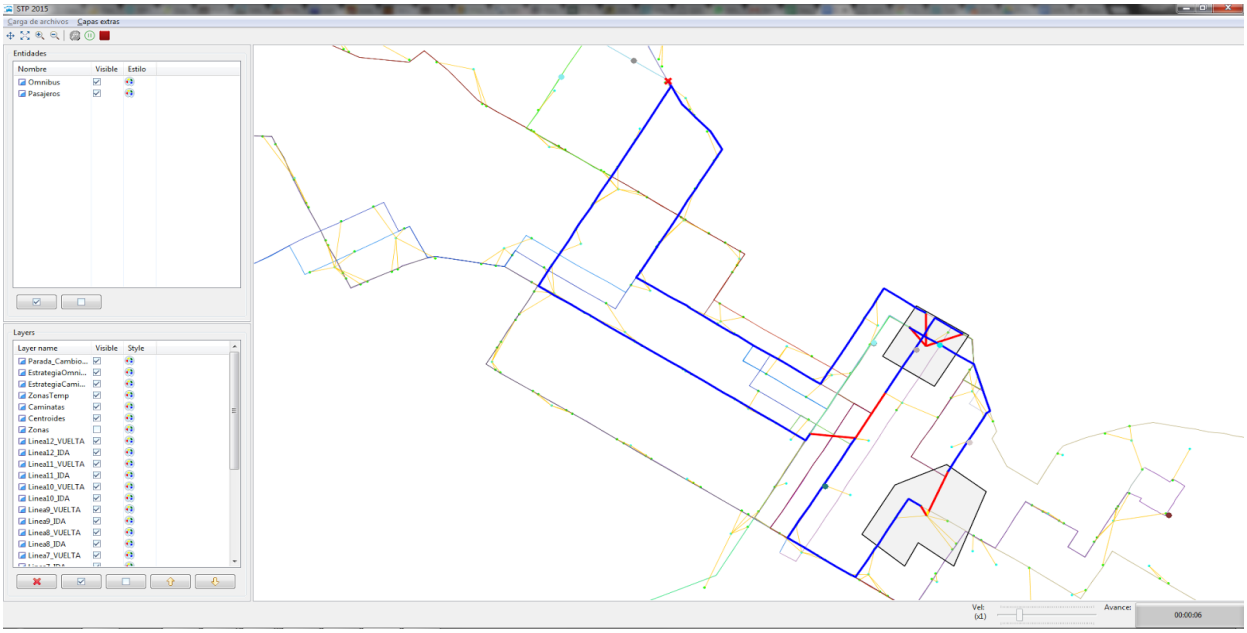
El SIG crea tres capas a partir de los datos recibidos: dos CapaLineas con nombres “EstrategiaCaminata” y “EstrategiaOmnibus” y una CapaPuntos con nombre “ParadaDeCambio”.

La capa *EstrategiaCaminata* representa cada caminata del pasajero seguido. Por las definiciones del sistema, la caminata o bien podría ser de centroide a parada, o de parada a parada. Dicha capa se completa con la información de los nodos de “***lis_centroide_parada***” y “***lis_parada_parada***”, que tienen identificadores de paradas y centroides. Por lo que el proceso para cargar la capa, basta con recorrer ambas listas y a partir de los ids de los puntos obtener su geometría, construir una entidad Línea entre cada par de puntos e ingresarla en la capa. Para el objetivo de la visualización es irrelevante el sentido de cada tramo ”dibujado”, es por ese motivo que no es necesario una lista de parada-centroide. Para destacar en la visualización a la capa se le definió un estilo simple de color rojo y más grueso que el resto.

La capa *EstrategiaOmnibus*, se sirve de “***lis_tomo_omnibus***” para cargar sus entidades y representa trayectos que el pasajero realiza sobre un ómnibus. Cada elemento de la lista es una 5-tupla, de estos valores se toman los primeros cuatro. A partir del *idLinea* y *tipoRecorrido* se puede obtener el recorrido que realiza el ómnibus, y con las paradas inicial y final se genera un sub recorrido. Con ese sub recorrido se crea una Línea para ingresar en la capa. La capa tiene un estilo simple de color azul e igual grosor de la capa anterior. El último valor de los elementos de “***lis_tomo_omnibus***” es una bandera booleana. Si está activada indica que la acción anterior fue tomar un ómnibus y bajar en la parada donde tomará el próximo ómnibus. Es una situación típica de trasbordo, donde un pasajero baja en una parada y en la misma debe de tomar otro ómnibus. A efectos de la visualización, esto se vería como una línea continua sin posibilidad de advertir que hay un punto de espera en una parada y además hay un cambio de líneas.

Para resolver el problema, es que está la tercera capa “*ParadaCambioParada*”, la cual agrega un Punto que coincide con la parada en cuestión. El estilo de la capa es un estilo simple que lo diferencia del resto de las entidades: una cruz roja, que hace notar que hay un punto particular en la estrategia que se está visualizando. Las tres capas creadas son cargadas en el “*MapContent*”, por lo que se puede modificar su estilo, ocultar o dejar visible.

La Figura 50 muestra la visualización de un plan de viaje con transbordo, donde el pasajero tiene la posibilidad de caminar desde el origen hacia 2 paradas, tomar un ómnibus y bajarse en su destino o tomar otro con más variantes.



▬ Viaje sobre ómnibus ▬ Caminata de pasajero **X** Descenso de ómnibus y ascenso en otro

Figura 50 - Plan de viaje

5.6.3 - Finalizar mostrar estrategia de pasajero a seguir

Descripción:

Finaliza el seguimiento del pasajero.

Actores:

visualizador y simulador.

Precondiciones:

El sistema debe estar iniciado y ejecutando. Existe un pasajero “P” que fue seleccionado a realizarle seguimiento.

Flujo Normal:

1. El caso de uso comienza cuando el simulador detecta que un pasajero “P” llega al final de su viaje.
2. El simulador informa al visualizador que debe eliminar los caminos de la estrategia del pasajero “P”.
3. El visualizador elimina los caminos de la estrategia del pasajero “P”.

Flujo Alternativo:

No hay

Poscondiciones:

No hay

Descripción de implementación

Cuando el simulador detecta que el pasajero a seguir termina su ciclo de vida, le informa al visualizador que el pasajero ya terminó su viaje. Por lo que el visualizador elimina del “MapContent” las capas mencionadas en la funcionalidad de la subsección *5.6.2 - Mostrar estrategia de pasajero a seguir*.

Capítulo 6 - Pruebas de la herramienta

En este capítulo se describen los distintos experimentos y casos de estudio que se simularon en la herramienta. En un principio se utiliza el modelo de una ciudad ficticia (Aldea), siendo un caso de pequeño porte por lo cual es ideal usarlo para la verificación de la implementación. Para validar la correctitud del modelo y el cumplimiento de los requerimientos de la herramienta se realizaron un conjunto de experimentos sobre las ciudades de Rivera y Montevideo. En cuanto a la validación del modelo para el caso de Rivera -el cual es de mediano porte-, se obtienen salidas (específicamente, resultados numéricos) que se comparan con resultados de trabajos anteriores. Para el caso de Montevideo -caso de gran porte-, no se cuenta con dicho caso implementado en las otras herramientas, por lo que no es posible comparar las salidas. Para este caso se comparará contra valores asumidos como razonables en base a conocimiento de la ciudad. En ambos casos, las salidas son el promedio de los tiempos de viaje de todos los pasajeros que llegan a su destino. El tiempo total de viaje es la suma de los tiempos de: la espera en el centroide origen; la caminata del centroide origen a la primer parada; la espera en la parada; el viaje en ómnibus (traslado); y la caminata de la parada final al centroide destino. Este conjunto de casos se utiliza además para mostrar que la herramienta soporta ciudades de diferente porte, requerimiento no funcional del proyecto.

En el *Anexo 5* se detalla el archivo de salida de la aplicación, el cual registra información de entidades y los momentos en los que interactúan con los diferentes eventos del sistema. A partir de esta información, se puede estudiar el correcto comportamiento de la simulación, como por ejemplo si el viaje de un pasajero es coherente con el ciclo de vida. En algunos experimentos descritos en este capítulo se utiliza esta información para su análisis.

6.1 - Verificación

El caso de la Aldea consiste de una ciudad que consta de tres zonas -cada una identificada por un centroide- y tres paradas -una por zona- conectadas por una caminata con su respectivo centroide. Sobre este modelo de ciudad se definen una serie de líneas de transporte público que permiten viajar entre las paradas. Las líneas son una circular y dos de ida y vuelta con cierta frecuencias. Originalmente la demanda de pasajeros es definida únicamente entre dos centroides.

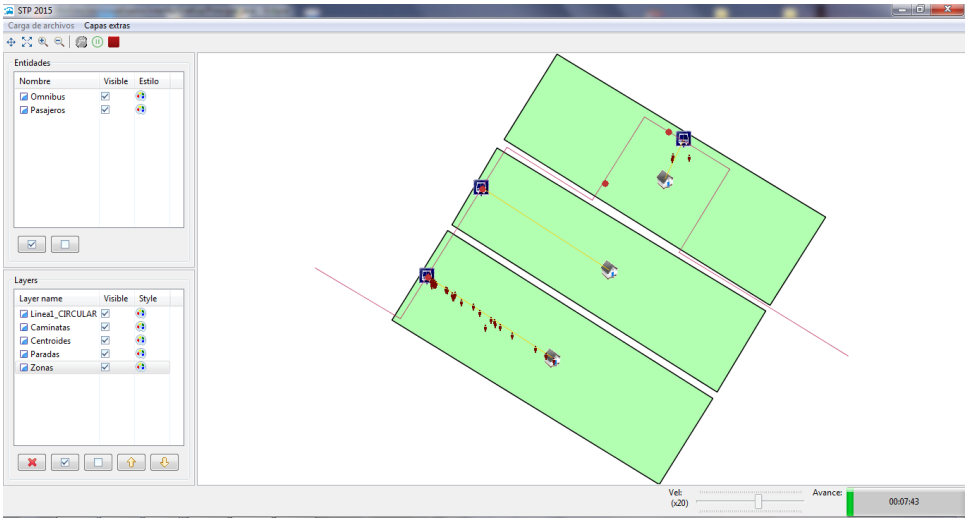


Figura 51 - Caso Aldea

Dado su pequeño porte, este caso permitió realizar y probar funcionalidades puntuales. Se utilizó para realizar pruebas tanto del simulador como de la visualización, además sirvió para verificar la integración entre ambos.

6.2 - Validación del modelo y de la herramienta

La validación del modelo se realiza a partir de resultados numéricos, en base a análisis de sensibilidad y a comparación con resultados de trabajos previos. Se realizan una serie de experimentos para validar el modelo y probar las funcionalidades de la herramienta desarrollada.

Para los experimentos, a partir del modelo base se definieron tres instancias del mismo, obtenidas mediante parametrizaciones específicas. En estas instancias se varía únicamente el sub modelo de comportamiento de los pasajeros. Los sub modelos de infraestructura y demanda, de líneas y recorridos de ómnibus, de aleatoriedad no tienen modificaciones.

Las instancias se definen según el comportamiento que tomen los pasajeros, dado por la estrategia:

- instancia 1: todos los pasajeros buscan Minimizar Caminata
- instancia 2: todos los pasajeros buscan Minimizar Tiempo de Viaje
- instancia 3: pasajeros pueden tomar una estrategia u otra con igual probabilidad

Se definen estas tres instancias del modelo con el fin de cubrir las variantes implementadas y poder realizar la validación de la herramienta desarrollada.

6.2.1 - Caso de estudio Rivera

Los experimentos que se simulan en el caso de Rivera se hacen con la siguiente configuración:

- Velocidad de pasajeros: 3 km/h
- Velocidad de ómnibus : 10 km/h
- Capacidad de ómnibus : 20 personas
- Tiempo de ómnibus en parada: 1 minuto
- Tiempo de Ejecución: 120 minutos
- Distribuciones desactivadas

Se emplearon los datos de Rivera tal como estaban disponibles desde proyectos anteriores: frecuencia de las líneas, demanda y frecuencia de arribo de los ómnibus a las paradas. Un ejemplo de ejecución se puede ver en la Figura 52 .

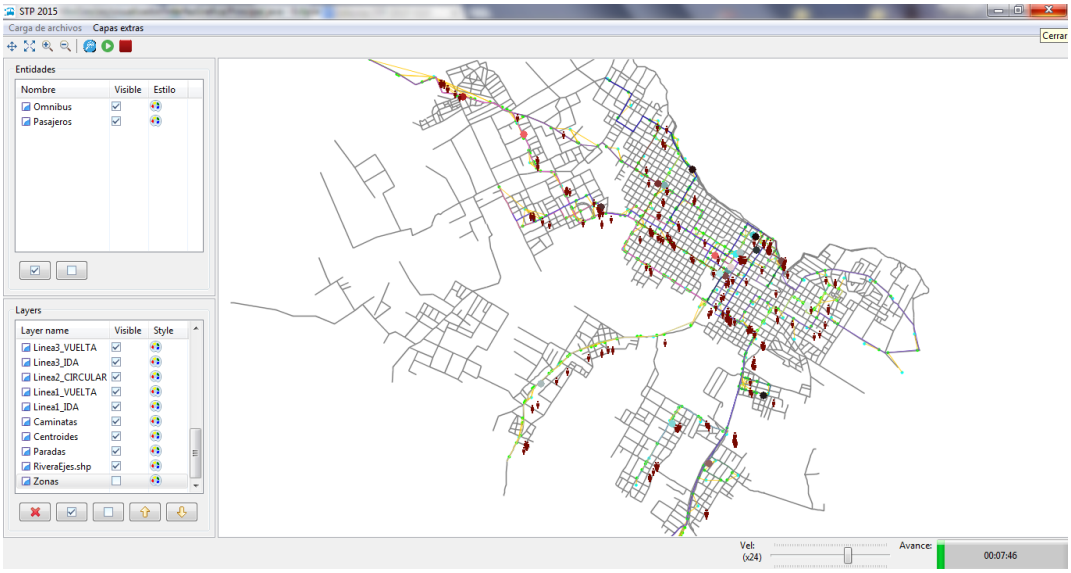


Figura 52 - Caso Rivera

El experimento 1 se usa como escenario principal para validar el modelo y las instancias del mismo. Los siguientes experimentos utilizan la instancia número 3. Estos sirven -además de para validar el modelo- para probar las distintas funcionalidades que brinda la herramienta. Las modificaciones a los parámetros que se realizan en cada experimento se hacen mediante el archivo de configuración.

Experimento 1 – Comparación con resultados previos

El experimento 1 es definido para validar el modelo. Consiste en comparar el promedio de tiempo total de viaje de los pasajeros con respecto a los modelos implementados en proyectos anteriores [1][6]. Dicho valor debe estar en el rango de 43 a 63 minutos.

El resultado de las ejecuciones y las características de las estrategias utilizadas por ejecución, se pueden observar en el Cuadro 1.

Número de instancia de modelo	Duración simulación (minutos)	Tiempo total de viaje (minutos)	Observaciones (cantidad de pasajeros que llegan a destino)
1	120	46,40	858
2	120	45,82	893
3	120	45,78	886
3	240	49,15	2121

Cuadro 1 - Tiempos de viaje y cantidad de observaciones según nro. de instancia y duración de simulación

Cómo se puede observar, en las primeras tres pruebas del experimento -instancias 1,2 y 3 con tiempo de ejecución 120 minutos- el promedio del tiempo total de viaje se encuentra en el rango definido. Dado que el tiempo de viaje promedio es mayor que la tercera parte del tiempo total de simulación, para la última prueba se duplica el tiempo de simulación. Se simula cuatro horas, observándose -que en proporción- mayor cantidad de pasajeros pudieron completar su viaje llegando a su centroide destino. El tiempo promedio de viaje de esta prueba también se encuentra en el rango establecido, aunque cabe acotar que el tiempo promedio de viaje aumenta tres minutos. Esto se puede dar debido a que una mayor duración permite que lleguen más pasajeros a destino -sobre todo los que demoran más tiempo-, y por lo tanto el cálculo es más preciso ya que se hace sobre más individuos.

Como el tiempo total de viaje para las tres instancias del modelo se encuentra dentro del rango establecido, se considera el modelo como validado.

Experimento 2 - Variar capacidad de los ómnibus de las líneas

El objetivo de este experimento es validar la funcionalidad que permite modificar la capacidad de los ómnibus para todas las líneas. El experimento consiste en realizar dos ejecuciones, una en la que los ómnibus tienen capacidad máxima diez pasajeros y otra con capacidad máxima veinte pasajeros. Se analizan -para ciertos ómnibus- cúal es la máxima cantidad de pasajeros que viajan simultáneamente en determinado momento en el ómnibus, verificando dicho valor contra la capacidad del ómnibus. Además se muestra el tiempo medio de viaje en el histograma "*tiempo de pasajero en el sistema*".

	Capacidad Máxima
cola de pasajeros de omnibus: 63	10
cola de pasajeros de omnibus: 64	10
cola de pasajeros de omnibus: 65	10
cola de pasajeros de omnibus: 66	10
cola de pasajeros de omnibus: 67	10

(A)

	Capacidad Máxima
cola de pasajeros de omnibus: 63	10
cola de pasajeros de omnibus: 64	11
cola de pasajeros de omnibus: 65	20
cola de pasajeros de omnibus: 66	15
cola de pasajeros de omnibus: 67	13

(B)

Figura 53 - Variación de capacidad de ómnibus

La Figura 53 (A) corresponde al primer prueba de este experimento, en el que la capacidad máxima del ómnibus es 10 pasajeros. En la Figura 53 (B) se busca verificar si es suficiente duplicar la capacidad de los

ómnibus para que estos no viajen en ningún momento con capacidad colmada. Por ejemplo para el ómnibus 63 se puede apreciar que la máxima cantidad de pasajeros sigue siendo 10, por lo que para ese ómnibus no sería necesario duplicar la capacidad. En cambio para el ómnibus 65, se puede apreciar que aunque se duplique la capacidad, en cierto momento transita con capacidad colmada.

Observando el reporte generado en la primer prueba, hay ciertos pasajeros que no podían realizar su viaje porque el ómnibus que debían tomar viajaba con capacidad colmada al pasar por su parada, al duplicarse la capacidad del ómnibus, estos pueden tomárselo. Esto explica la pequeña disminución del tiempo total de viaje de la prueba 2 respecto a la prueba 1 y el aumento de arribos a destino (Observaciones), como se puede observar en el Cuadro 2.

Prueba	Tiempo total de viaje (minutos)	Observaciones
1 - (Capacidad 10)	46,24	723
2 - (Capacidad 20)	45,78	886

Cuadro 2 - Impacto en la modificación de la capacidad de los ómnibus

Experimento 3 - Variar velocidad de los ómnibus de las líneas

El objetivo de este experimento es validar la funcionalidad que permite modificar la velocidad de las líneas. Se realizan dos pruebas con diferentes velocidades. A partir de esta modificación se analizaron los tiempos de finalización de los recorridos para algún caso en particular. Además se muestra el tiempo medio de viaje en el histograma "*tiempo de pasajero en el sistema*" con el objetivo de ver cómo influye esta variación en los pasajeros. A partir de la traza se analizaron los tiempos de inicio y fin de recorrido de un ómnibus en particular, en este caso correspondiente a la línea 8. Para ambas pruebas el ómnibus se crea en tiempo 3600 segundos, como se puede observar en la Figura 54 .

'Evento_Generador_Omnibus#15'	Omnibus_Sim#66 Se crea el omnibus de la linea 8 de tipo de recorrido: Vuelta en tiempo 3600.0
	schedules 'Evento_Arribo_Omnibus_Parada#962' of 'Omnibus_Sim#66' at 3600
	schedules 'Evento_Generador_Omnibus#15' at 4800

Figura 54 - Generación de ómnibus de la línea 8

En la Figura 55 se observa la prueba en que la velocidad de la línea es 10 km/h. El ómnibus arriba a la parada destino en tiempo 6733 segundos (112 minutos).

6733 'Evento_Arribo_Omnibus_Parada#1679'	'Omnibus_Sim#66'	Omnibus_Sim#66 El omnibus llega a la parada 155
		schedules 'Evento_Retiro_Parada_Omnibus#1682' of itself at 6733
'Evento_Retiro_Parada_Omnibus#1682'		Omnibus_Sim#66 Omnibus se retira de la parada 155
		Información Parada: 155
		Largo Cola de Omnibus: 0
		Omnibus_Sim#66 Omnibus arribo a la parada destino del recorrido en tiempo 6733.0

Figura 55 - Tiempos ómnibus con velocidad 10 km/h

En la Figura 56 se observa la prueba en que la velocidad de la línea es 20 km/h. El ómnibus arriba a la parada destino en tiempo 5620 segundos (94 minutos).

```
5620 'Evento_Arribo_Omnibus_Parada#1564'    'Omnibus_Sim#66'    Omnibus_Sim#66 El omnibus llega a la parada 155
                                                               schedules 'Evento_Retiro_Parada_Omnibus#1569' of itself at
                                                               5620
     'Evento_Retiro_Parada_Omnibus#1569'                       Omnibus_Sim#66 Omnibus se retira de la parada 155
                                                               Información Parada: 155
                                                               Largo Cola de Omnibus: 0
                                                               Omnibus_Sim#66 Omnibus arribo a la parada destino del
                                                               recorrido en tiempo 5620.0
```

Figura 56 - Tiempos ómnibus con velocidad 20 km/h

Cómo era de esperar, duplicar la velocidad de los ómnibus disminuye considerablemente el tiempo del recorrido, lo cual hace que el tiempo total de viaje de los pasajeros también disminuya de forma considerable. En el Cuadro 3 se observa el impacto en el tiempo promedio de viaje de los pasajeros.

Prueba	Tiempo total de viaje (minutos)	Observaciones
Velocidad 10 km/h	45,78	886
Velocidad 20 km/h	36,04	1002

Cuadro 3 - Impacto del cambio de velocidad de los ómnibus en el viaje de los pasajeros

Experimento 4 - Variar velocidad de caminata de los pasajeros

Este experimento consiste en modificar la velocidad de caminata de los pasajeros, para poder validar la funcionalidad y ver cómo impacta la variación de velocidad en los tiempos de los pasajeros. Además a modo de resumen se muestra el tiempo medio de viaje en el histograma "*tiempo de pasajero en el sistema*".

La primer prueba utilizada es con velocidad de caminata de los pasajeros 3 km/h, mientras que en la segunda prueba se aumenta la velocidad en 50%, resultando en una velocidad de caminata 4,5 km/h para todos los pasajeros. En la Figura 57 se puede observar un pasajero en la primer prueba, el cual se crea en el centroide número 58 a los 425 segundos de iniciada la simulación. Este arriba al centroide destino en tiempo 2653 segundos.

```
425  'Evento_Generador_Pasajero#221'          ----              Pasajero_Sim#290 Se crea el pasajero en tiempo 425.0
                                                                schedules 'Evento_Arribo_Pasajero_Centroide#437' of
                                                                'Pasajero_Sim#290' at 425
     'Evento_Arribo_Pasajero_Centroide#437'   'Pasajero_Sim#290'  Pasajero_Sim#290 Arriba a centroide 58
                                                                Pasajero_Sim#290 Debe caminar a parada 222
2653 'Evento_Arribo_Pasajero_Centroide#1030'  'Pasajero_Sim#290'  Pasajero_Sim#290 Pasajero llega a centroide en tiempo 2653.0
                                                                después de haber viajado
```

Figura 57 - Tiempos pasajero con velocidad caminata 3 km/h

En la Figura 58 , se muestra el resultado de la prueba dos, para el mismo pasajero de la primer prueba pero con velocidad de caminata un 50% mayor (4,5 km/h). Tambíen se crea en el tiempo 425 segundos en el centroide número 58. Pero a raíz del aumento en su velocidad de caminata arriba al centroide destino en tiempo 2540 segundos.

```
425   'Evento_Generador_Pasajero#221'          ----                  Pasajero_Sim#290 Se crea el pasajero en tiempo 425.0
                                                                     schedules 'Evento_Arribo_Pasajero_Centroide#443' of
                                                                     'Pasajero_Sim#290' at 425
                                                                     schedules 'Evento_Generador_Pasajero#221' at 850
      'Evento_Arribo_Pasajero_Centroide#443'   'Pasajero_Sim#290'     Pasajero_Sim#290 Arriba a centroide 58
                                                                     Pasajero_Sim#290 Debe caminar a parada 222
2540  'Evento_Arribo_Pasajero_Centroide#993'   'Pasajero_Sim#290'     Pasajero_Sim#290 Pasajero llega a centroide en tiempo 2540.0
                                                                     después de haber viajado
```

Figura 58 - Tiempos pasajero con velocidad caminata 4,5 km/h

Vemos que la diferencia en los tiempos de caminata para este pasajero en particular, impacta en su tiempo total de viaje, disminuyendo casi dos minutos (113 segundos).

A modo resumen, aumentar la velocidad de caminata en un 50% para todos los pasajeros no impacta significativamente el tiempo promedio del total de viaje de los pasajeros. En el Cuadro 4 observaremos dicho impacto.

Prueba	Tiempo total de viaje (minutos)	Observaciones
1 - (Velocidad 3 km/h)	45,78	886
2 - (Velocidad 4,5 km/h)	44,29	901

Cuadro 4 - Impacto de la velocidad de caminata de los pasajeros en su tiempo de viaje

Experimento 5 - Variar tiempos de subida y/o bajada de los pasajeros de los ómnibus

El objetivo de este experimento es validar la funcionalidad que permite modificar el tiempo que toman los pasajeros en subir y/o bajar del ómnibus. Cabe recordar que este tiempo es igual para todos los pasajeros y se asume que coincide con el tiempo que permanece el ómnibus en la parada en caso de que haya pasajeros para subir o bajar. Se verifica cómo impacta este cambio en algún pasajero en particular. Además, a modo de resumen se muestra el tiempo medio de viaje en el histograma "*tiempo de pasajero en el sistema*". En la Figura 59 se puede observar un pasajero en la prueba base (pasajeros toman un minuto en subir o bajar del ómnibus), el cual se crea en el tiempo 425 segundos en el centroide número 58. Este arriba al centroide destino en tiempo 2653 segundos.

```
425   'Evento_Generador_Pasajero#221'          ----                  Pasajero_Sim#290 Se crea el pasajero en tiempo 425.0
                                                                     schedules 'Evento_Arribo_Pasajero_Centroide#437' of
                                                                     'Pasajero_Sim#290' at 425
                                                                     schedules 'Evento_Generador_Pasajero#221' at 850
      'Evento_Arribo_Pasajero_Centroide#437'   'Pasajero_Sim#290'     Pasajero_Sim#290 Arriba a centroide 58
                                                                     Pasajero_Sim#290 Debe caminar a parada 222
2653  'Evento_Arribo_Pasajero_Centroide#1030'  'Pasajero_Sim#290'     Pasajero_Sim#290 Pasajero llega a centroide en tiempo 2653.0
                                                                     después de haber viajado
```

Figura 59 - Tiempo de viaje de pasajero para demora de un minuto en subir o bajar del ómnibus

En cambio cuando los pasajeros toman dos minutos para ascender o descender, el pasajero que estamos analizando llega al centroide destino casi 4 minutos más tarde. Siendo el tiempo de llagar a destino de 2881 segundos, cuando antes lo hacía en 2653 segundos. Lo cual se puede observar en la Figura 60 .

```
425  'Evento_Generador_Pasajero#221'          ----                  Pasajero_Sim#290 Se crea el pasajero en tiempo 425.0
                                                                    schedules 'Evento_Arribo_Pasajero_Centroide#437' of
                                                                    'Pasajero_Sim#290' at 425
                                                                    schedules 'Evento_Generador_Pasajero#221' at 850
     'Evento_Arribo_Pasajero_Centroide#437'  'Pasajero_Sim#290'      Pasajero_Sim#290 Arriba a centroide 58
                                                                    Pasajero_Sim#290 Debe caminar a parada 222
2881 'Evento_Arribo_Pasajero_Centroide#1070' 'Pasajero_Sim#290'     Pasajero_Sim#290 Pasajero llega a centroide en tiempo 2881.0
                                                                    después de haber viajado
```

Figura 60 - Tiempo de viaje de pasajero para demora de dos minutos en subir o bajar del ómnibus

Tal como era predecible, el aumentar el tiempo que toman los pasajeros para subir y/o bajar de un ómnibus impacta de manera significativa el tiempo promedio del total de viaje de los pasajeros. Ya que en cada parada que un ómnibus debe detenerse, esté lo hará el doble de tiempo. En el Cuadro 5 observaremos dicho impacto.

Prueba	Tiempo total de viaje (minutos)	Observaciones
1 minuto	45,78	886
2 minutos	51,90	809

Cuadro 5 - Impacto de la demora en subir o bajar del ómnibus en el tiempo de viaje

Experimento 6 - Varias replicaciones

Para finalizar con el estudio del caso de Rivera, se ejecutaron 30 replicaciones para la instancia número 3 del modelo, con las distribuciones activadas. El objetivo de este experimento es realizar varias corridas independientes para tomar varias muestras como respuestas, tanto para calcular la media como la desviación estándar de los tiempos promedio de viaje de los pasajeros.
La configuración para esta prueba se da por las siguientes variables:

- Velocidad de pasajeros: 3 km/h, desviación estándar 0,6
- Velocidad de ómnibus : 10 km/h, desviación estándar 2
- Capacidad de ómnibus : 20 personas
- Tiempo de ómnibus en parada: 1 minuto
- Tiempo de Ejecución: 120 minutos
- Distribuciones activadas

En el Cuadro 6 se puede observar que la media de los tiempos promedio total de viaje de los pasajeros -para las treinta replicaciones- es de 2833 segundos (48 minutos), y que la desviación estándar es de 39,71 segundos. Este resultado sigue siendo consistente con los resultados numéricos de estudios anteriores y los resultados del experimento 1.

Histograma (Tiempo de viaje de pasajeros)	Cantidad de Observaciones	Promedio	Desviación estándar
Treinta replicaciones	30	2833,18636	39,71517

Cuadro 6 - Datos de treinta replicaciones Rivera

6.2.2 - Caso de estudio Montevideo

Este es un caso de mediano/gran porte, compuesto por datos de líneas, zonas y paradas de Montevideo que fueron sometidas a simplificaciones y transformaciones que se detallan en la sección *4.3 - Montevideo*. Este escenario tiene en común varias de las características de Rivera. Las línea de ómnibus son de tipo ida,

vuelta, o circular; teniendo una frecuencia de 30 minutos entre salidas de ómnibus por recorrido. Dichas líneas definen una cantidad de 278 recorridos, mientras que hay un total de 4706 paradas interconectadas con eventualmente varias líneas. A su vez se cuenta con 317 zonas bien delimitadas que abarcan toda la superficie -con un promedio de 1,68 km2 por zona- y cada una define su centroide. Para la matriz de demanda, no se cuenta con datos y (por razones de alcance del proyecto) no es el objetivo replicar exactamente el caso. Por este motivo se construye una matriz cuadrada de tamaño 317 x 317 que cuenta con 1480 lugares no nulos generados aleatoriamente. Esto es, pares de centroides para los cuales hay demanda de pasajeros.

Los sistemas desarrollados al momento por el grupo de investigación no soportan la simulación de este caso de estudio, por lo que nos es imposible comparar tiempos contra simuladores anteriores. Para demostrar la validez de la simulación se toma como referencia un valor razonable de tiempo de viaje dado el conocimiento de la ciudad. Además, este caso -junto a los de las ciudades de Rivera y la Aldea- sirve para demostrar que la herramienta desarrollada, a partir de simples modificaciones en los datos de entrada recabados, soporta la ejecución de cualquier ciudad con características similares a los casos nombrados.

El caso de Montevideo se simula utilizando la instancia número 3 del modelo -los pasajeros pueden tomar una estrategia u otra con igual probabilidad-, con la siguiente configuración de variables:

- Velocidad de pasajeros: 3 km/h, desviación estándar 0,6
- Velocidad de ómnibus: 10 km/h, desviación estándar 2
- Capacidad de ómnibus: 20 personas
- Tiempo de ómnibus en parada: 1 minuto
- Tiempo de Ejecución: 240 minutos
- Distribuciones activadas

Una vez obtenidos los datos necesarios, se procede a la ejecución del caso. En la Figura 61 se puede observar una captura de la salida visual.

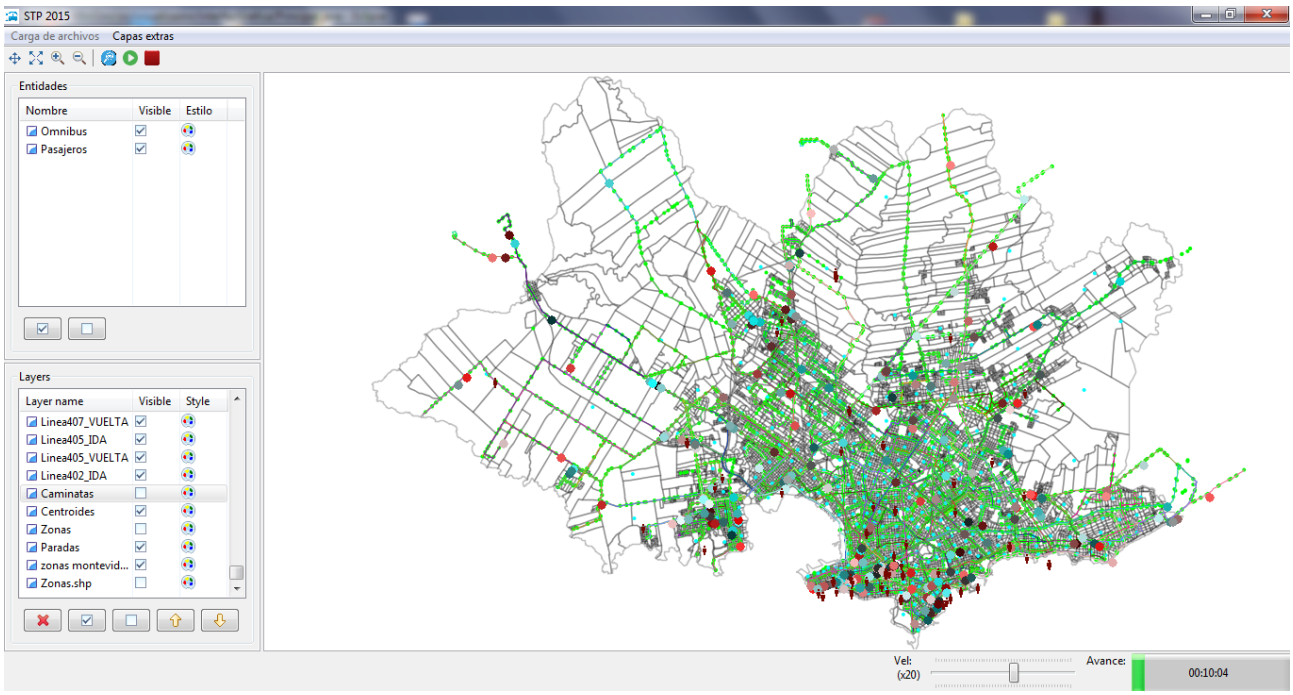


Figura 61 - Caso Montevideo

La simulación de treinta replicaciones para el caso de Montevideo da como resultado lo expuesto en el Cuadro 7. Se puede observar que la media de los tiempos promedio total de viaje de los pasajeros -para las treinta replicaciones- es de 5219 segundos (87 minutos), y que la desviación estándar es de 48,63 segun-

dos. Según los datos ingresados para Montevideo, podemos hacer una aproximación gruesa de los valores esperados de las salidas, concretamente:

- El tiempo promedio de espera en centroide se estima en 8 minutos, ya que la mitad de los pasajeros tienen espera 0 minutos -debido a la estrategia de menor caminata- y la otra mitad entre 0 y 30 minutos.
- El tiempo promedio de caminatas está en el entorno de los 13 minutos, ya que la velocidad promedio de las personas es de 3 km/h y se calcula una media de 0,7 km por caminata -según la zonificación elegida-. Por cada viaje hay dos caminatas.
- El tiempo promedio de espera en una parada se estima en 12 minutos, ya que la mitad de los pasajeros calculan llegar en un tiempo cercano a que pase el ómnibus -cercano a 0 minutos de espera para la estrategia de menor tiempo de viaje esperando en el centroide el mayor tiempo posible-, mientras que la otra mitad estaría entre los 0 y los 30 minutos. Se aumenta un poco la estimación ya que hay pasajeros que pierden el ómnibus o estos pasan llenos -no pudiendo subir al que tenían planificado-, lo que hace que esa espera sea mayor.
- Con los valores obtenidos en los puntos anteriores y el tiempo promedio de viaje de la ejecución; el tiempo promedio de traslado en ómnibus estaría cercano a los 41 minutos, lo que consideramos -según los conocimientos de los recorridos habituales de la ciudad de Montevideo- está dentro de los valores esperados, siendo igualmente la estimación más gruesa.

Por lo antes expuesto, podemos ver que los valores obtenidos de la ejecución de la aplicación son razonables con los valores esperados, validando este caso de estudio.

Histograma (Tiempo de viaje de pasajeros)	Cantidad de Observaciones	Promedio	Desviación estándar
Treinta replicaciones	30	5219,92901	48,63614

Cuadro 7 - Datos de treinta replicaciones Montevideo

6.2.3 - Performance

La ejecución de las pruebas utiliza la configuración de componentes físicos especificados en el Cuadro 8.

Componente	Descripción
CPU	Intel(R) Core(TM) i7-2670 2.2Ghz
Memoria RAM	8 Gb
Sistema Operativo	Windows 10 home

Cuadro 8 - Hardware utilizado

En el Cuadro 9 se detalla el tiempo de ejecución de los casos que se hicieron treinta replicaciones, para estos la visualización se encuentra desactivada. Para aquellos casos en que la visualización se encuentre activa, el tiempo de ejecución depende directamente de la velocidad a la que se desee ver la representación visual.

Caso	Tiempo (segundos)
Rivera - 30 replicaciones	25
Montevideo - 30 replicaciones	180

Cuadro 9 - Tiempos de ejecución de la aplicación

Se observa que la aplicación tiene tiempos de respuesta aceptables tanto para casos de pequeño como de gran porte.

6.3 - Cumplimiento de otras funcionalidades

En las secciones *6.1* y *6.2*, se han presentado resultados de experimentos que permiten validar la herramienta. En esta sección se valida el requerimiento funcional: seguimiento a un pasajero, y el requerimiento no funcional: soporte de implementación de distintas estrategias con cierto grado de complejidad, además de la correcta representación visual de estas últimas. Se corrobora el funcionamiento de estrategias que contengan transbordos y/o varios caminos para realizar un viaje. La complejidad de la estrategia dependerá de las necesidades y el desarrollo del implementador, pero se debe hacer explícito la validación de que el sistema soporta su visualización y funcionamiento. Para esto, se implementan estrategias arbitrarias, que no siguen un criterio en particular más que asignar líneas a tomar, paradas donde subir o bajar, transbordos y trayectos a seguir, de tal forma de generar una estrategia compleja. La aplicación debe ser funcional a las estrategias, y el resultado de las mismas se debe visualizar correctamente. Para esto se consideran dos escenarios para la validación, Rivera y Montevideo, ya que la aplicación debe ser capaz de comportarse correctamente en ambos casos. En cada caso se presentan las estrategias implementadas en un esquema y luego se ejecuta la aplicación para comparar su visualización con el esquema inicial con el fin de validar la visualización de seguimiento a un pasajero.

Para validar el aspecto funcional de la aplicación -con la nueva estrategia implementada- se adjunta un extracto de la traza generada que describe el comportamiento de un pasajero dado por la estrategia. La validación se hará efectiva si la traza es coherente con el esquema inicial.

6.3.1 - Caso Rivera - Estrategia compleja

En este escenario se tomaron los datos de Rivera y se les realizó ciertas modificaciones para permitir crear una estrategia lo suficientemente compleja para el propósito buscado. En la Figura 62 , se expone un esquema que describe el viaje que propone la estrategia implementada para el caso de Rivera.

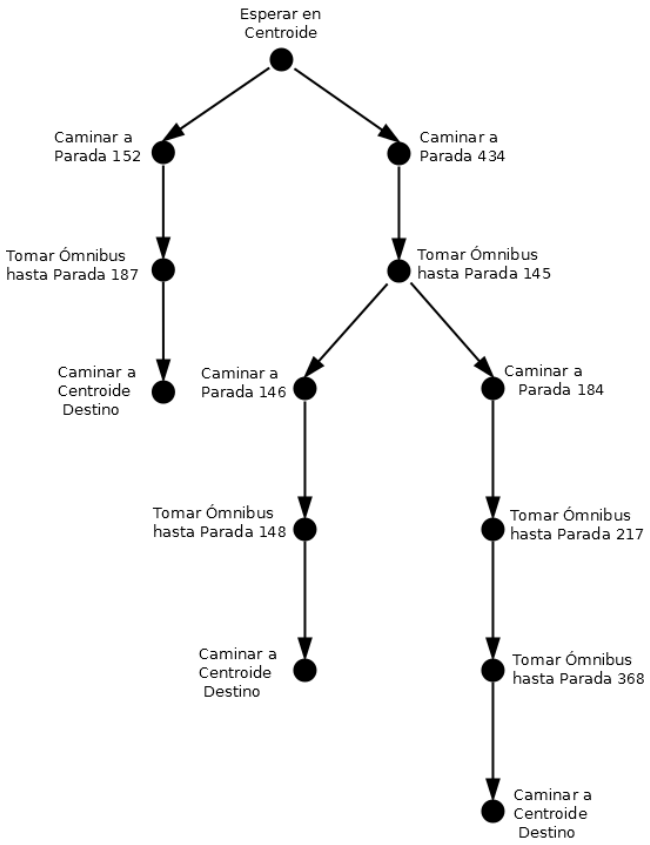


Figura 62 - Esquema de la estrategia compleja de Rivera

En la Figura 63 se muestra el sistema ejecutando con la implementación de la estrategia antes mencionada, y con la funcionalidad de seguimiento de pasajero activa. Cabe acotar que la figura contiene información adicional a la salida del sistema (identificadores de paradas, de tramos y caminatas) con el fin de esclarecer su explicación. El pasajero en el centroide origen (zona ZO) tiene dos opciones de caminatas (c1 y c3) hacia las paradas (p152 o p434) respectivamente. Si toma la caminata c1 llega a la parada p152, luego toma el ómnibus que lo conduce por el tramo t1 hasta la parada p187, y luego a través de la caminata c2 llega al centroide destino (zona ZD). Otra opción, desde el centroide origen, el pasajero camina por c3 y llega a p434, luego toma el ómnibus que lo conduce por t2 hasta descender en la parada p145. Aquí hay dos opciones de caminatas (c4 y c6), Si toma c4, hasta la parada p146, luego aborda el ómnibus que lo conduce por t3 hasta la parada p148, para llegar al centroide destino a través de la caminata c5. Si toma la caminata c6 llega a la parada p184, luego toma el ómnibus por el tramo t4, hasta descender en la p217, donde toma el ómnibus que lo lleva por el tramo t5 hasta la parada p368, donde a través de la caminata c7 llega al centroide destino(zona ZD) .

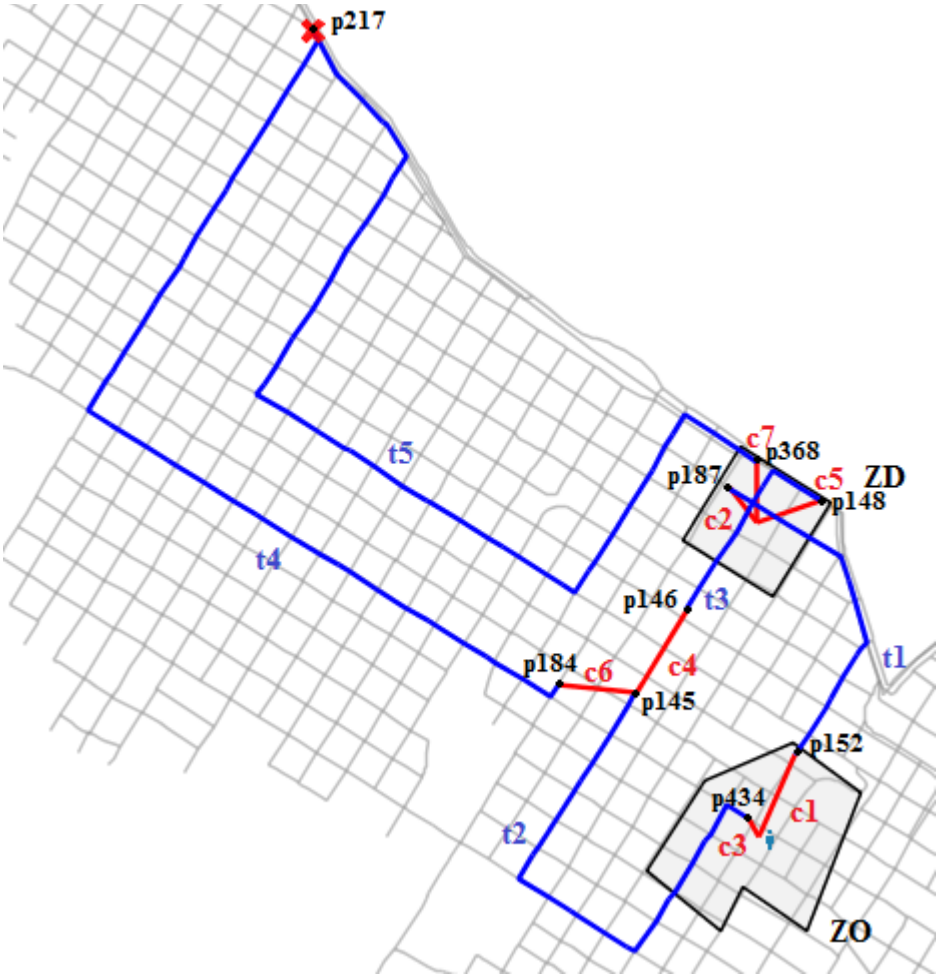


Figura 63 - Representación de la estrategia compleja de Rivera

Dado que el análisis de la visualización es consistente con el esquema de la estrategia implementada se da por validada la funcionalidad seguimiento a un pasajero en el caso Rivera.

Resta validar si la aplicación se comportó conforme a la nueva implementación. Para ello, en la Figura 64 se muestra la traza de un pasajero que tiene la implementación de la estrategia compleja. En la traza se puede observar detalladamente el comportamiento del pasajero en la aplicación y la consistencia tanto con el esquema inicial, como con la funcionalidad seguimiento de un pasajero. Por lo que también se da por vali-

dado que la aplicación sea funcional a la implementación de nuevas estrategias, incluso con cierto grado de complejidad en el caso de Rivera.

0	'Evento_Generador_Pasajero#1'		Pasajero_Sim#1 Se crea el pasajero en tiempo 0.0
	'Evento_Arribo_Pasajero_Centroide#1'	'Pasajero_Sim#1'	Pasajero_Sim#1 Arriba a centroide 63
			Pasajero_Sim#1 Debe esperar en centroide 63 tiempo 0.1
6	'Evento_Arribo_Pasajero_Centroide#2'	'Pasajero_Sim#1'	Pasajero_Sim#1 Debe caminar a parada 434, tiempo de caminata 2.0
126	'Evento_Arribo_Pasajero_Parada#1'	'Pasajero_Sim#1'	Pasajero_Sim#1 Pasajero llega a parada 434
			Pasajero_Sim#1 Debe tomar omnibus, linea 2 tipo CIRCULAR, parada destino 145
2685	'Evento_Arribo_Omnibus_Parada#674'	'Omnibus_Sim#3'	Omnibus_Sim#3 El omnibus llega a la parada 434
			Omnibus_Sim#3 Suben 1 pasajeros
2745	'Evento_Pasajero_Sube_a_Omnibus#1'	'Omnibus_Sim#3' and 'Pasajero_Sim#1'	Pasajero_Sim#1 Pasajero 1 se subió al omnibus 3 de la linea 2
3410	'Evento_Arribo_Pasajero_Parada#2'	'Pasajero_Sim#1'	Pasajero_Sim#1 Pasajero llega a parada 145
			Pasajero_Sim#1 Debe caminar a parada 146
3470	'Evento_Arribo_Pasajero_Parada#3'	'Pasajero_Sim#1'	Pasajero_Sim#1 Pasajero llega a parada 146
			Pasajero_Sim#1 Debe tomar omnibus, linea 12 tipo VUELTA, parada destino 148
5338	'Evento_Pasajero_Sube_a_Omnibus#2'	'Omnibus_Sim#62' and 'Pasajero_Sim#1'	Pasajero_Sim#1 Pasajero 1 se subió al omnibus 62 de la linea 12
5686	'Evento_Arribo_Pasajero_Parada#4'	'Pasajero_Sim#1'	Pasajero_Sim#1 Pasajero llega a parada 148
			Pasajero_Sim#1 Debe caminar a centroide 66
5746	'Evento_Arribo_Pasajero_Centroide#3'	'Pasajero_Sim#1'	Pasajero_Sim#1 Pasajero llega a centroide en tiempo 5746.0 después de haber viajado

Figura 64 - Traza de pasajero con estrategia compleja Rivera

6.3.2 - Caso de Montevideo - Estrategia compleja

Como en la subsección *6.3.1*, es de interés en esta sección validar la funcionalidad “seguimiento de un pasajero”, y verificar que la aplicación soporta la implementación de nuevas estrategias con cierto grado de complejidad, para un caso de gran porte cómo Montevideo.

En la Figura 65 , se puede observar el esquema que describe el viaje que propone la estrategia compleja implementada para la ciudad de Montevideo.

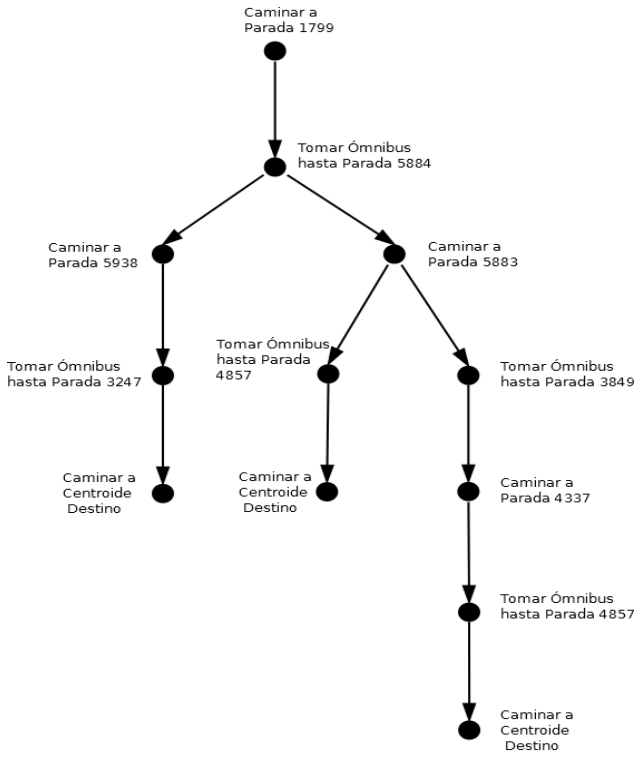


Figura 65 - Esquema de estrategia compleja para Montevideo

En la Figura 66 se muestra el sistema ejecutando con la implementación de estrategia antes mencionada, y con la funcionalidad de seguimiento de pasajero activa. Cabe acotar que la figura contiene información adicional a la salida del sistema (identificadores de paradas, de tramos y caminatas) con el fin de esclarecer su explicación. El pasajero que se encuentra en el centroide origen tiene una opción de caminata (c1) hacia la parada p1799. Luego toma el ómnibus que lo conduce por el tramo t1 hasta la parada p5884. Al arribar a dicha parada, tiene dos opciones de caminata (c2 y c3), una hacia la parada p5938 y otra hacia la parada p5883. Si toma la caminata c2, luego toma el ómnibus que lo conduce por t2 hasta descender en la parada p3247, para llegar al centroide destino a través de la caminata c4. En cambio, si toma la caminata c3 tiene dos opciones para tomar ómnibus, una por el tramo t3 con parada destino p4057 y otra por el tramo t4 con parada destino p3049. Si toma la opción del tramo t3, al arribar a la parada p4057 debe caminar al centroide destino a través de la caminata c5. Si toma la opción del tramo t4, al arribar a la parada p3849, debe caminar a través de la caminata c6 a la parada p4337, para luego tomar el ómnibus que lo lleva para el tramo t5 a la parada destino p4857. Al arribar a dicha parada debe caminar al centroide destino a través de la caminata c7.

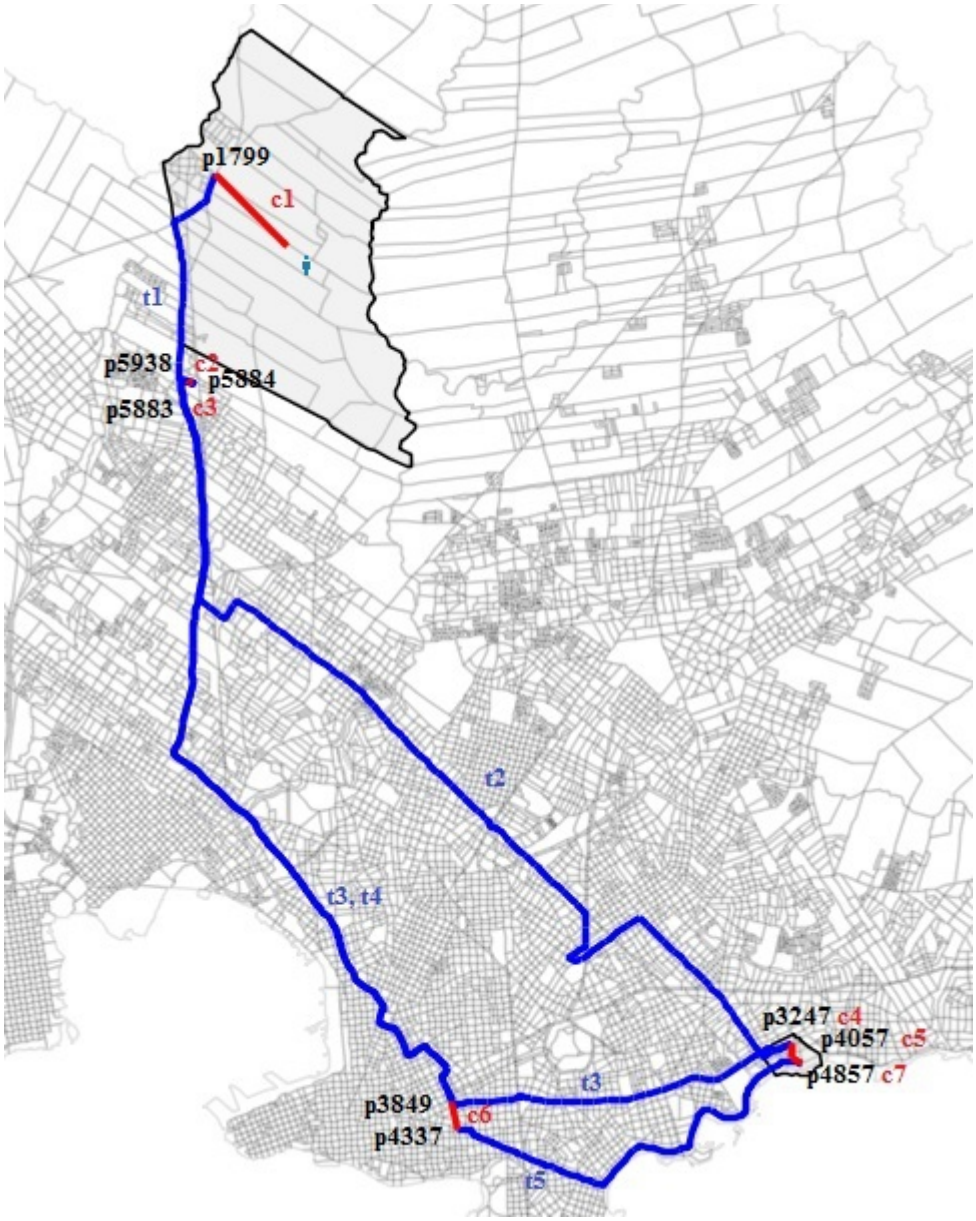


Figura 66 - Representación de la estrategia compleja de Montevideo

Dado que el análisis de la visualización es consistente con el esquema de la estrategia implementada se da por validada la funcionalidad seguimiento a un pasajero en el caso de Montevideo.

Resta validar si la aplicación se comporta conforme a la nueva implementación. Para ello, a continuación se incluye un extracto de la traza de un pasajero que tiene asignada la nueva estrategia.

0	'Evento_Generador_Pasajero#1'		Pasajero_Sim#1 Se crea el pasajero en tiempo 0.0
	'Evento_Arribo_Pasajero_Centroide#1'	'Pasajero_Sim#1'	Pasajero_Sim#1 Arriba a centroide 66
			Pasajero_Sim#1 Debe caminar a parada 1799
			Pasajero_Sim#1 Debe caminar a parada 1799, tiempo de caminata 2.0
120	'Evento_Arribo_Pasajero_Parada#1'	'Pasajero_Sim#1'	Pasajero_Sim#1 Pasajero llega a parada 1799
			Pasajero_Sim#1 Debe tomar omnibus, linea 141 tipo IDA, parada destino 5884
613	'Evento_Arribo_Omnibus_Parada#1424'	'Omnibus_Sim#277'	Omnibus_Sim#277 El omnibus llega a la parada 1799
			Omnibus_Sim#277 Suben 3 pasajeros
673	'Evento_Pasajero_Sube_a_Omnibus#1'	'Omnibus_Sim#277' and 'Pasajero_Sim#1'	Pasajero_Sim#1 Pasajero 1 se subió al omnibus 277 de la linea 141
2199	'Evento_Arribo_Pasajero_Parada#11'	'Pasajero_Sim#1'	Pasajero_Sim#1 Pasajero llega a parada 5884
			Pasajero_Sim#1 Debe caminar a parada 5938
2259	'Evento_Arribo_Pasajero_Parada#14'	'Pasajero_Sim#1'	Pasajero_Sim#1 Pasajero llega a parada 5938
			Pasajero_Sim#1 Debe tomar omnibus, linea 116 tipo IDA, parada destino 3247
3471	'Evento_Pasajero_Sube_a_Omnibus#11'	'Omnibus_Sim#507' and 'Pasajero_Sim#1'	Pasajero_Sim#1 Pasajero 1 se subió al omnibus 507 de la linea 116
10331	'Evento_Arribo_Pasajero_Parada#175'	'Pasajero_Sim#1'	Pasajero_Sim#1 Pasajero llega a parada 3247
			Pasajero_Sim#1 Debe caminar a centroide 141
10391	'Evento_Arribo_Pasajero_Centroide#71'	'Pasajero_Sim#1'	Pasajero_Sim#1 Pasajero llega a centroide en tiempo 10391.0 después de haber viajado

Figura 67 - Traza de pasajero con estrategia compleja Montevideo

La Figura 67 muestra la traza de un pasajero que tiene la implementación de la estrategia compleja. En la traza se puede observar detalladamente el comportamiento del pasajero en la aplicación y la consistencia tanto con el esquema inicial, como con la funcionalidad seguimiento de un pasajero. Por lo que también se da por validado que el sistema sea funcional a la implementación de nuevas estrategias, incluso con cierto grado de complejidad en el caso de Montevideo.

Dado que se implementaron dos estrategias complejas, una para el caso de Rivera y otra para el caso de Montevideo, y en ambos experimentos se pudo validar tanto la funcionalidad de seguimiento de pasajero como que la aplicación es funcional a ambas estrategias, entonces se da por validada dicha funcionalidad y el requerimiento no funcional de que la aplicación debe soportar distintos estrategias incluso con cierto grado de complejidad.

Capítulo 7 - Conclusión y trabajos a futuro

En este capítulo se presentan las conclusiones y los trabajos a futuro.

7.1 - Conclusiones

Una vez concluido el trabajo realizado, se puede afirmar que se cumplió con los objetivos del proyecto. Se creó una única herramienta para la simulación y visualización de casos de estudio sobre modelos de sistemas de transporte público. Dicha herramienta incluye funcionalidades que contribuyen a la tarea de validar y experimentar con los modelos. Dentro de la experimentación con la aplicación desarrollada, cabe destacar la posibilidad de interacción con el usuario en tiempo de simulación, permitiendo por ejemplo seleccionar un pasajero y hacer un seguimiento del mismo. Además es posible pausar, aumentar o disminuir la velocidad de simulación. Dicha interacción es un punto muy relevante para el usuario, ya que permite el análisis espacial a través de la visualización y la posibilidad de incidir en la simulación.

La herramienta desarrollada soporta la simulación y visualización de ciudades de diferentes portes, como lo son Rivera y Montevideo. El ciclo de vida impactado en el modelo y el diseño de la herramienta permite distintos comportamientos de pasajeros, lo cual da a lugar a futuros estudios sobre otros modelos de interés.

Con respecto a los datos de entrada se construyó un proceso que transforma datos que no cumplen los requerimientos de la aplicación a datos que sí los cumplen. El contar con el conocimiento interno de la aplicación desarrollada permite saber cuáles, cómo y qué relaciones guardan algunas características de los datos de entrada. Por lo que el proceso construye dichas relaciones, permitiendo no ser tan estricto en las exigencias con los datos de entrada. También se pudo corroborar que el proceso puede ser utilizado con insumos de datos de ciudades con diferentes características, pero que cumplen con los requisitos solicitados.

Para poder desarrollar la aplicación se utilizó un modelo base, que también sirvió para comprobar que la misma sirve para validar modelos. Se validó el modelo base comparando resultados de ejecución con salidas obtenidas en proyectos anteriores. También se estudió el comportamiento del sistema al variar algunos parámetros de ejecución, como por ejemplo la capacidad del ómnibus, las velocidades de los ómnibus y de los pasajeros. Se pudo comprobar parte del correcto funcionamiento de la herramienta, siendo los valores obtenidos dentro de lo lógico para su comportamiento. Para probar que el simulador funciona según lo planificado es que se utilizó el caso la Aldea. Al ser un caso de pequeño porte se puede tener un control de la acción de cada componente, y de esta forma se corroboró el correcto funcionamiento.

También se realizó un caso de estudio sobre la ciudad de Montevideo. En él se puede confirmar el correcto funcionamiento del sistema para ciudades de mediano/gran porte. Obteniendo resultados acordes a los conocimientos de la ciudad.

Para poder verificar otras funcionalidades se realizaron algunos estudios específicos. Se crearon estrategias particulares con cierto grado de dificultad, con ellas se pudo validar la visualización del seguimiento de un pasajero, además se pudo comprobar que el sistema soporta que los viajes de los pasajeros contengan trasbordos y agregar estrategias complejas -como por ejemplo las explicadas en [4]-, funcionalidad deseada para poder realizar nuevos y más completos experimentos.

En cuanto a la gestión del proyecto, se sufrió algunos contratiempos que generó un corrimiento en el cronograma establecido. Ejemplo de ello, son los requerimientos. Estos se fueron refinando con el transcurso del proyecto. Una primera aproximación, fue que la solución se construya a partir de la modificación de las herramientas disponibles en el grupo de investigación, adaptándolas a los objetivos del proyecto. La incompatibilidad en tecnologías, como ser las herramientas de simulación, limitaciones en los desarrollos anteriores para alcanzar los nuevos objetivos y lo engorroso de incorporarse a un desarrollo ajeno, hicieron que esta idea fuera descartada. Motivo de retraso fue también la ejecución de un caso de gran porte como Montevideo, ya que aunque la intención era solo ver que la herramienta soporte un caso como el mencionado, se incorporó como objetivo y se procedió con su configuración, experimentación y validación. Las tecno-

logías también representaron un reto, ya que el conocimiento que se tenía no era suficiente para determinar su potencial y cuales eran más adecuadas para soluciones específicas. El abordaje de este punto fue a través de estudio y experimentación, muchas veces generando prototipos o pruebas particulares para mitigar riesgos futuros en el desarrollo del proyecto.

En cuanto al desarrollo de la aplicación se realizó en Java, tanto el visualizador y el simulador. Esta elección no es arbitraria, ya que es coherente con las bibliotecas usadas, además de ser un lenguaje conocido para los integrantes del equipo.

Luego de finalizado el proyecto se valoraron decisiones adoptadas en el transcurso del proyecto.

En la etapa inicial del mismo se decidió realizar un proceso de transformación de datos independiente del desarrollo de la aplicación. Incluso los desarrollos se solaparon. El proceso toma datos y los transforma a las necesidades de la aplicación. Luego de finalizado el proyecto y haber ganado un bagaje importante en las herramientas utilizadas, vemos que una alternativa a dicha decisión hubiese sido integrar el proceso de transformación a la aplicación desarrollada, ya que el potencial de la herramienta utilizada en su construcción así lo permite.

Analizando cada uno de estos puntos concluimos que luego de estudio y trabajo se logró una comprensión acabada del problema que permitió generar una solución que cumple con todos los objetivos y resultados esperados establecidos.

7.2 - Trabajos a futuro

En el proceso de desarrollo del proyecto se detectaron funcionalidades de utilidad para el usuario, que pueden contribuir aún más a la tarea de validación y experimentación. Estos puntos no fueron tomados en cuenta por encontrarse fuera del alcance definido del proyecto. Igualmente en este apartado se listan como contribución a eventuales trabajos futuros.

A continuación se presentan dichos puntos organizados por componente.

Simulador:

- Permitir por medio del archivo de configuración cambiar la capacidad o velocidad de una línea en particular. Estos parámetros se encuentran implementados a nivel de línea, pero al momento se cargan desde el archivo de configuración, con el mismo valor para para todas las líneas.
- El tiempo que se detiene un ómnibus en una parada es un valor fijo, configurable por parámetro. Este valor se podría incluir en el modelo por medio de una distribución de probabilidades. Actualmente, este valor no se considera en el cálculo de los tiempos de las estrategias, pero sí influye en los resultados.
- Permitir la personalización de los reportes de salida adecuándose a las necesidades particulares de cada proyecto específico.

Visualización:

- Posibilidad de cambiar la estrategia de un pasajero en cualquier momento de la ejecución. El presente desarrollo le asigna una estrategia a un pasajero en función de los parámetros ingresados, dicha estrategia se mantiene en el ciclo de vida de la aplicación o del pasajero. Este punto propone crear una funcionalidad que otorgue la posibilidad al usuario de cambiar la estrategia de un pasajero en tiempo de ejecución.
- Perfeccionar la captura de datos para el seguimiento de un pasajero: actualmente el sistema, aunque permite la selección gráfica (a través de eventos de selección con el mouse) de zonas origen y destino del viaje de un pasajero, no se recomienda su uso, ya que no actualiza correctamente la interfaz gráfica del visualizador, pudiendo generar gráficos confusos. Perfeccionar este mecanismo incluiría una correcta actualización de los gráfico del Visualizador. Actualmente, el sistema permite la

selección de zonas por identificador.

- Incluir el proceso de transformación de datos dentro del SIG. Actualmente, primero se realiza un proceso de normalización de datos a través de scripts con *gvSIG*. En un segundo paso se ejecuta el sistema. Con esta funcionalidad se centraliza la solución en una única herramienta, el sistema solución, pudiendo así prescindir de *gvSIG* y del conocimiento de scripting para dar soporte a los scripts.
- Generar archivos shapefile de las capas que son generadas por la aplicación, como son la de visualización de seguimiento de un pasajero o las caminatas de los mismos. Dichas capas se generan a nivel de memoria. Generando dichos archivos se permite conservar los datos más allá de la herramienta y eventualmente realizar estudios sobre ellos o utilizarlos como una capa más de modo comparativo.
- Mostrar información en función de las necesidades del usuario. Para los siguientes casos, los valores de las variables mencionadas están disponibles en el simulador o en la estructura, en caso de que no estén, pueden ser fácilmente calculables. A continuación se lista:
  - Permitir poder seleccionar un pasajero y ver detalles como:
    - identificador
    - estado del pasajero (espera, caminando, viajando)
    - centroides origen y destino
    - estrategia asignada
    - líneas a las que espera
  - Permitir poder seleccionar un ómnibus y ver detalles como:
    - línea
    - tipo de recorrido
    - capacidad actual, capacidad máxima
    - tiempo en llegar a destino
    - tiempo próxima arribo a parada
    - próxima salida de la línea
  - Permitir poder seleccionar un centroide y ver detalles como:
    - cuántos pasajeros lo tienen como origen
    - frecuencia promedio de creación de pasajeros
    - centroides a los que se llegan desde él
    - tiempo promedio de viaje de los pasajeros que salen desde este centroide
  - Permitir poder seleccionar una parada y ver detalles como:
    - identificador de paradas
    - líneas que paran en ella
    - cantidad de pasajeros esperando (cantidad de elementos en cola)
    - cantidad de ómnibus detenidos

Anexo 1 - Decisiones sobre herramienta de transformación de datos

Para la transformación de datos se tiene que tomar la decisión de qué herramienta SIG, y en particular cuáles de sus módulos de scripting, se ajusta mejor a las necesidades del presente proyecto.

Se consideraron las herramientas de software libre SIG, no comerciales, de mayor reputación y que además cuenten con documentación apropiada y actualizada de scripting. Se seleccionaron dos de acuerdo a sus prestaciones: gvSIG y QGIS [14][15].

Luego, en base a estudio y experimentación en el área se decidió por una. A continuación se describen los puntos que se tuvieron en cuenta para la selección.

Se observó que QGIS posee un módulo de scripting que incluye editor y consola. El editor proporciona las herramientas típicas de un editor de texto aunque algo disminuido. gvSIG también posee editor y consola aunque el editor es muy limitado, careciendo incluso de la mayoría de operaciones básicas.

gvSIG utiliza como lenguaje de desarrollo Jython, una adaptación que permite ejecutar programación de Python con un programa que trabaja sobre el lenguaje de Java.

QGIS usa como lenguaje de scripting PyQGIS: es el conjunto de bindings que el proyecto QGIS aporta para que el desarrollador acceda a las bibliotecas de QGIS (que están escritas en C++) a través de Python.

Se observó, en cuanto a la documentación, que se encuentran documentos conceptualmente más completos de Jython que de PyQGIS, además que se encontraron más sitios web vivos en el mismo sentido.

La experimentación constó de dos tipos de pruebas:

- De complejidad del lenguaje y funcionalidades: aquí se desarrollaron pequeños script que crean shapes, acceden a capas, modifican atributos, y acceden a la geometría.
  El resultado fue satisfactorio para gvSIG y no tanto para QGIS, ya que pudieron desarrollar scripts semejantes con un esfuerzo mucho menor Jython que PyQGIS.
- De performance: se implementó un script que toma para cada entidad de una capa un atributo, y actualiza otro con ese valor. La capa tiene aproximadamente 60.000 entidades y cada entidad tenía 30 atributos de tipos disímiles.
  El resultado fue concluyente, la ejecución de gvSIG se interrumpió en muchas ocasiones, mientras que en QGIS se efectuó con éxito sin inconvenientes.

Para arribar a una conclusión, si bien se tomó en cuenta cada punto anterior se le dió distinta relevancia a cada uno. La conclusión fue que de los resultados de la experimentación QGIS fue más performante y robusto pero con un costo de aprendizaje demasiado alto en comparación al que presenta gvSIG, mientras que ésta última, muestra algunos signos de degradación e inestabilidad para volúmenes muy grandes de datos que escapan a los que se esperan procesar en este y en proyectos similares.

A su vez, la documentación de gvSIG es más precisa y abundante, y se observa una mayor actividad en la web. Por lo que la elección fue gvSIG y su módulo de scripting.

Anexo 2 - Elección de la herramienta SIG y pruebas funcionales.

Elección de la herramienta

Para poder elegir una herramienta SIG para la visualización del sistema, se toman en cuenta las siguientes características:

- Que sea de fácil manipulación y modificación.
- Que sea libre, que no necesite una licencia para su uso.
- Que esté implementado en un lenguaje conocido.
- Que tenga documentación y soporte adecuados.
- Que pueda integrarse fácilmente con el simulador.
- En lo posible que sea de escritorio y offline.
- Que soporte un número importante de elementos visuales.

A partir de estas cualidades, se investigan las siguientes herramientas:

- ArcGIS
- QGIS
- GVSig
- GeoTools
- Visor PublicTransport

ArcGIS

ArcGIS es un conjunto de productos desarrollados por ESRI. Entre estos productos se investigaron dos de ellos: ArcGIS for Desktop y ArcGIS for Developers [16].

ArcGIS for Desktop es una herramienta ya desarrollada para la manipulación de datos geográficos, y por lo tanto genérica. Esto hace que no sea conveniente para su utilización en un sistema específico.

ArcGIS for Developers es un conjunto de API's para diferentes lenguajes de programación, como por ejemplo Android, iOS, .Net, Java. Se muestran las principales ventajas y desventajas encontradas para su utilización.

Ventajas

- Es una de las herramientas más utilizadas.
- Se tiene conocimiento previo de su utilización.
- Es de fácil manipulación y modificación.
- Puede ser utilizado en diferentes lenguajes de programación.
- Tiene mucha documentación y soporte.

Desventajas

- Necesita conexión a internet. Si bien es posible realizar una herramienta de escritorio, ArcGIS se basa en webservices para partes básicas de su funcionamiento.
- No es totalmente libre. Tiene una licencia libre pero es acotada, al utilizar alguna funcionalidad requerida para la manipulación de capas, no es posible usarla con esta licencia.
- Su última actualización de las bibliotecas Java fue en diciembre de 2014.

Si bien ArcGIS tiene muchas ventajas para su utilización, el que no sea libre para todas sus funcionalidades y que tenga que tener conexión a internet para su utilización, hace que no sea conveniente aplicar una solución con esta herramienta.

QGis

Ventajas

- Está desarrollado en un lenguaje conocido como lo es C++.
- Utiliza Scripts Python.
- Es Open Source.
- Se tienen conocimientos de la herramienta.

Desventajas

- Es una herramienta de procesamiento de capas.
- Es de uso general y no específico.
- Difícil para su modificación.

GVSig

Ventajas

- Está desarrollado en un lenguaje conocido como lo es Java.
- Utiliza Scripts Python.
- Es Open Source.

Desventajas

- Es una herramienta de procesamiento de capas.
- Es de uso general y no específico.
- Difícil para su modificación.

GeoTools

GeoTools es una biblioteca Java que proporciona herramientas para datos geoespaciales [17].

Ventajas

- Está desarrollado en un lenguaje conocido como lo es Java.
- Tiene buena documentación.
- Es Open Source.
- Al ser una biblioteca, se puede desarrollar un sistema nuevo.

Desventajas

- No tiene una GUI soportado para la visualización. Tiene prototipos desarrollados por usuarios avanzados pero no están soportados por el proyecto.

Visor PublicTransport

Herramienta utilizada en los proyectos anteriores para la visualización de la salida del simulador [6].

Ventajas

- Herramienta desarrollada en proyectos anteriores.
- Es Open Source.
- Está desarrollado en un lenguaje conocido.

Desventajas

- Si bien está desarrollado en un lenguaje conocido, no es el de más conocimiento, por lo que la curva de aprendizaje del lenguaje puede influir negativamente.
- Como la herramienta está desarrollada para una entrada de datos final, la adaptación a un sistema en tiempo real puede llegar a ser más complicado.

Una vez investigado las diferentes opciones, la utilización de la herramienta GeoTools cumple con los requisitos solicitados. El lenguaje de programación utilizado es el más conocido, Java. A su vez, este lenguaje es común al resto de los componentes de la aplicación, lo que facilita su comunicación

Pruebas funcionales

Luego de elegida la herramienta, se procede a realizar diferentes pruebas para ver la viabilidad de su utilización.

Swing vs SWT

Uno de los problemas con los que cuenta GeoTools, es que el proyecto en sí no cuenta un una interfaz de usuario propia. Esto hace que la documentación sobre el mismo sea escasa y poco amigable.

Igualmente, existen dos GUI desarrolladas por usuarios avanzados investigadas para su utilización. Ellas son GT-Swing y GT-SWT desarrolladas en Swing y SWT respectivamente.

Las principales diferencias se basan en la visualización ya que Swing tiene su propia visual y SWT obtiene la visual y funciones del sistema operativo en el cual se ejecuta. Como contraparte Swing necesita la misma librería para la ejecución en cualquier sistema operativo, mientras que SWT necesita una librería diferente para cada uno.

En las Figura 68 y Figura 69 se puede ver las diferencias de la visual.

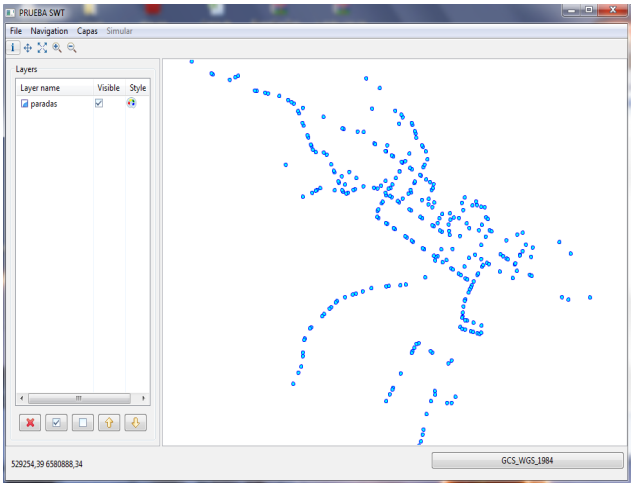


Figura 68 - Visual de GeoTools con Swing

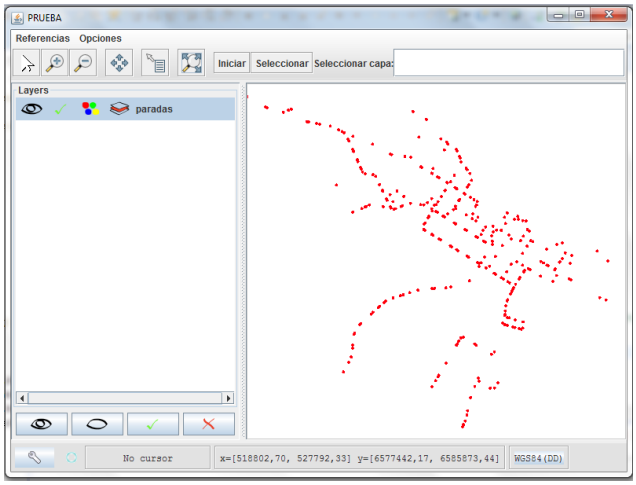


Figura 69 - Visual de GeoTools con SWT

Como no era un requerimiento que funcione en todos los sistemas operativos y alcanza con que funcione en Windows 10 de 64 bits, y que SWT es más amigable en funcionalidad y visual, ya que se adapta al sistema operativo, se decide utilizar este.

Caso de prueba

Para poder comprobar la viabilidad de la utilización del GeoTools, se realizó un pequeño caso de prueba. En él se carga un mapa y a partir de un evento de presionar el botón izquierdo del mouse, se dibujan recorridos aleatorios sobre el mapa.

El funcionamiento de esta prueba es el siguiente:

1. Se carga un mapa desde un archivo shapefile.
2. Al seleccionar un punto sobre el mapa, se genera un recorrido aleatorio desde ese punto con cincuenta puntos de largo, de distancia fija entre un punto y el siguiente.
3. Se dibuja el recorrido en el mapa.
4. Se multiplica por tres la cantidad de recorridos para la próxima selección de punto en el mapa.
5. Los recorridos generados no se olvidan, para redibujar en cada nuevo evento de repintar el mapa.

Con esta prueba se pueden validar dos puntos, la manipulación de la representación visual según la necesidad y la cantidad de puntos a dibujar en cada momento.
En las Figura 70 , Figura 71 , Figura 72 , Figura 73 y Figura 74 se muestra la ejecución antes de seleccionar un punto del mapa y durante cuatro selección de puntos.

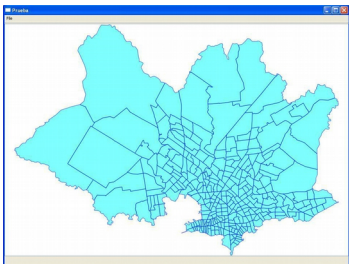


Figura 70 - Inicial

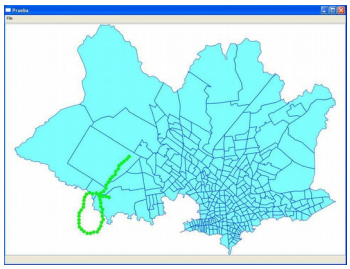


Figura 71 - Primer evento

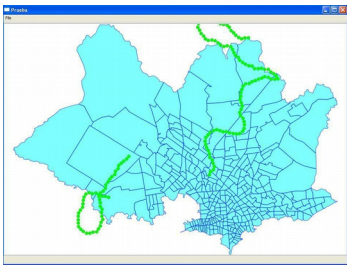


Figura 72 - Segundo evento

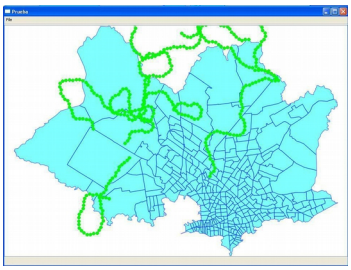


Figura 73 - Tercer evento

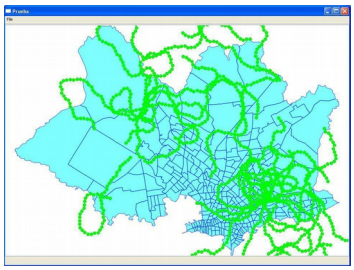


Figura 74 - Cuarto evento

Con las pruebas realizadas se comprueba que la herramienta GeoTools es adecuada para el fin del proyecto.

Anexo 3 - Elección de biblioteca para simulación a eventos discretos y pruebas funcionales.

Elección de la herramienta

Para poder elegir un simulador, se tomó en cuenta las siguientes cualidades:

- Que sea de fácil manipulación y modificación.
- Que sea libre, que no necesite una licencia para su uso.
- Que esté implementado en un lenguaje conocido.
- Que tenga documentación y soporte adecuados.
- Que se pueda integrar fácilmente con el SIG.

A partir de estas cualidades, se investigan las siguientes herramientas:

- EOSimulator
- DESMO-J

EOSimulator

EOSimulator es un framework orientado a objetos para el desarrollo de simulaciones. Fue diseñado e implementado en el Departamento de Investigación Operativa de la Facultad de Ingeniería de la Universidad de la República [18].

Ventajas

- es open source.
- diseñado para ser extensible y modificable por los propios usuarios.
- soporta el enfoque de eventos, tanto en dos o tres fases.

Desventajas

- último release data del año 2006.
- desarrollado en C++, lo que implica un mayor esfuerzo para comunicación con el SIG elegido.
- no cuenta con reportes de salida ni logs.

Cabe acotar que la biblioteca también soporta animación 2D, se desarrolla mostrando las entidades y datos claves utilizando íconos y símbolos.

DESMO-J

DESMO-J es un framework orientado a objetos dirigido al desarrollo de modelos de simulación. El acrónimo "DESMO-J" es sinónimo de "simulación de eventos discretos y Modelización en Java". Permite la construcción rápida y flexible de modelos de simulación de eventos discretos en Java, soportando los paradigmas orientado a eventos (2 fases) y orientados a procesos. DESMO-J fue desarrollado por el departamento de Ciencias de la Computación de la Universidad de Hamburgo. La primer versión data del año 1993, la biblioteca se continúa desarrollando hasta la fecha en términos de proyectos SourceForge [19].

Ventajas

- es open source.
- es portable ya que está desarrollado en Java.
- se actualiza en forma regular (la versión con la que se trabajó es la 2.4.2 de fecha Noviembre 2014, al momento existe una nueva versión 2.5.0 publicada en Noviembre del año 2015. La nueva versión presenta una mejora en la eficiencia de la ejecución de los procesos).
- salida gráfica en formato html.
- buena documentación y tutorial online en la página del proyecto.

Desventajas

- maneja el enfoque de dos fases pero no el enfoque de tres fases.
- desconocimiento de la biblioteca.

Cabe acotar que la biblioteca también soporta animación 2D y visualización 3D. La animación 2D se desarrolla mostrando las entidades y datos claves utilizando íconos y símbolos. La visualización 3D está basada en JAVA3D.

Prueba final con DESMO-J

Para confirmar el simulador elegido se utiliza el tutorial definido en el sitio web del simulador. Este es un tutorial paso a paso sobre cómo construir un modelo de simulación orientado a eventos con DESMO-J. Comienza con una descripción del modelo, luego orienta a diseñarlo, identificando las entidades pertinentes y definir el comportamiento de las mismas mediante los eventos. Luego de tener en mente las entidades y los eventos, comienza la implementación del sistema. El modelado de un sistema en DESMO-J, utilizando el enfoque de eventos, se realiza implementando clases derivadas de Model, Event, Entity, y SimProcess. Luego se variaron ciertos parámetros en nuestro modelo con el fin de hacer pruebas más profundas y conocer la herramienta. Finalmente, se observan los resultados generados vía archivos html.Por el curso de Simulación a Eventos Discretos se conoce el simulador EOSimulator, por lo que no se realizó una prueba en específica en el marco de este proyecto.

La prueba final con DESMO-J permite de manera sencilla y ágil desarrollar un modelo de simulación, gracias a que el desarrollo de este consiste en un desarrollo Java y que las interfaces a utilizar son sumamente claras y se encuentran documentadas. Luego de desarrollado el modelo, se ajustan los parámetros para los reportes, los cuales son de gran utilidad.

Por las ventajas comentadas y dado que el Sistema de Información Geográfica elegido debe ser compatible con Java, la biblioteca de simulación elegida es DESMO-J.

Anexo 4 - Datos para el caso de estudio de Montevideo

Para realizar un caso de estudio, es necesario contar con los datos de entrada normalizados según la especificación de la aplicación. Como el proceso de transformación de datos se basa en los datos obtenidos de la ciudad de Rivera, es necesario comprobar que es aplicable para distintas ciudades. Por lo que se investigó con datos de la ciudad de Montevideo.

En AGESIC[6] se encuentran publicados datos abiertos de diferentes organismos [20]. Es allí donde se obtuvieron la mayoría de los archivos necesarios para poder realizar este estudio.

Los información obtenida es:

- **v_uptu_paradas.shp**: Contiene la información de todas las paradas de Montevideo.
  Descripción del shape v_uptu_paradas: Capa de puntos.
  - Cod_ubic_p Código de la ubicación de parada.
  - Cod_varian Código de la variante de línea de ómnibus (refiere v_uptu_lsv).
  - Ordinal Número correlativo de la parada en el trayecto de la variante.
  - Calle Nombre de la calle sobre la que se ubica la parada.
  - Cod_calle1 Código de la Calle según nomenclátor oficial de Montevideo.
  - Esquina Nombre de la esquina más próxima .
  - Cod_calle1 Código de la Esquina según nomenclátor oficial de Montevideo.
  - X Coordenada X de la ubicación (SIRGAS2000 UTM 21s).
  - Y Coordenada Y de la ubicación (SIRGAS2000 UTM 21s).

- **v_uptu_lsv .shp:** Contiene la información de las líneas de ómnibus de Montevideo.
  Descripción del shape v_uptu_lsv: Capa de líneas.
  - Gid Código identificador de uso interno.
  - Cod_linea Código de la línea de transporte.
  - Desc_linea Descripción de la línea de transporte (ej: 145, D10, etc).
  - Ordinal_su Número correlativo de la sublinea en la línea.
  - Cod_sublin Código de la sublínea de recorrido.
  - Desc_subli Descripción de la sublínea de recorrido (ej: ADUANA-PORTONES).
  - Cod_varian Código de la variante de recorrido (para vincular con v_uptu_parada).
  - Desc_varia Descripción de la variante (Se asigna sentido A, a todas las líneas que circulan con destino hacia la zona sur del departamento (Ciudad Vieja, Centro, Cordón, P. Rodó, P. Carretas, Pocitos, Buceo, Malvín), además de todas las líneas que atraviesan esta zona en sentido Oeste-Este. Se asigna sentido B, a todas las líneas en sentido contrario al antes mencionado.

La construcción de una zonificación escapa al alcance del proyecto, por lo que se consideró relevante utilizar una zonificación existente. Para este caso se obtuvo un shapefile de zonas que utiliza el Correo Uruguayo y que está publicado en la página del INE[7] [21].

- **MontevideoZP.shp:** Es una capa de polígonos de una división en zonas postales de la ciudad. Las zonas postales están definidas tal que el tamaño de cada zona es inversamente proporcional a la densidad de su población, es decir que cuanto menos habitantes por polígono mayor es su área, lo que permite aumentar las paradas y recorridos en zonas donde son escasos. Esto cumple con la mayoría de los criterios de zonificaciones de los estudios de transporte [22]. Otro aspecto a resaltar

6 Agencia de Gobierno Electrónico y Sociedad de la Información y del Conocimiento, dependiente de Presidencia de la República

7 Instituto Nacional de Estadística

es que los polígonos no son superpuestos y tienen atributo Id único. El shapefile tiene otros atributos que no serán mencionados.

El formato de los datos encontrados distan mucho de los datos base de la ciudad de Rivera. Es por ese motivo que se realiza un pre-proceso para adaptar la información recogida, para luego ejecutar los scripts necesarios y normalizarlos según las necesidades.

A continuación se detalla el pre-proceso ejecutado, para cada uno de los archivos shapefile necesarios para la entrada de datos.

Zonas

En este caso, como se obtuvo un único archivo shapefile de polígonos, no se realizó ningún pre-proceso. O sea que se utiliza para la transformación de datos de entrada el shapefile “MontevideoZP.shp”.

Paradas

El archivo obtenido de paradas (v_uptu_paradas.shp), si bien es un shapefile de puntos donde se ubica cada parada, cada punto está repetido tantas veces como recorridos de ómnibus pasan sobre la parada. Por ello se realizó una depuración de las paradas “repetidas” para obtener un punto por parada. Vale aclarar que se llamaremos puntos repetidos a dos o más entidades que compartan su geometría, aunque pueden o no, diferir en el restos de sus atributos. En este caso el pre-proceso constó en la ejecución de la herramienta SAGA y su función: Remove Duplicate Points [23].

Dicha herramienta, detecta puntos repetidos, mantiene el primero encontrado (arbitrariamente) y elimina el resto en el archivo de salida. Además si así se configura, se puede asociar el número de puntos repetidos a la única entidad con geometría en común que quedará en la capa resultante. Esta última práctica es útil para controlar la exactitud del proceso, no tiene mayor aporte para el presente proyecto.

Este preproceso concluyó con la creación de los identificadores de la capa, creando un nuevo campo: Id_Parada, y copiando en este, el valor del campo Cod_ubic_p.

Líneas de Omnibus

Los datos obtenidos de las líneas de ómnibus, son los que más se diferencian entre Montevideo y Rivera. Para Montevideo ya vienen todas las líneas en un único archivo shape. Por lo que simplemente se le agregan los atributos necesarios para la carga de datos:

- ID: Se podría utilizar el Gid que viene en el shape, pero se decidió generar un nuevo identificador.
- ID_LINEA: Se copia el valor de Desc_linea.
- TIPO: Si Desc_varia tiene el valor “A” se asigna el tipo “IDA” , si el valor es “B” se asigna tipo “VUELTA”. En caso de que para una misma Desc_linea no exista un valor “B”, esto es un recorrido de tipo “CIRCULAR”.

Las líneas de ómnibus de Montevideo tienen más de dos recorridos en la mayoría de los casos. Esto se debe a que cada línea puede tener recorridos completos, cortos, nocturnos, etc. Para solo tener a lo máximo dos recorridos por cada línea (esto es parte de lo requerido por el sistema), se dejaron los recorridos que tienen el menor Cod_sublin.

Algo que no incluye las líneas de Montevideo, es por las paradas que pasa cada recorrido. Para saber cuales son las paradas de cada recorrido, el archivo de paradas v_uptu_paradas.shp contiene la información necesaria. Cada recorrido tiene un valor único de código de variante. Con este valor en el shapefile de paradas, se puede saber cuáles son las paradas por las que pasa el recorrido. Para saber en qué orden realizan esas paradas es que se utiliza el campo Ordinal.

Con esto se genera un archivo ***Recorridos.csv*** que contiene los campos ID_LINEA correspondiente a cada recorrido a utilizar, y PARADAS donde están, para cada recorrido, las paradas ordenadas separadas por coma.

Anexo 5 - Salidas del simulador

Un experimento en DESMO-J automáticamente produce ciertos archivos con formato preestablecido, el más importante de ellos es el Reporte, el cual brinda un reporte completo de la corrida. Para la validación y/o calibración del modelo son útiles los archivos Error, Debug y Traza.

El archivo de Traza permite obtener una buena visión del comportamiento del modelo. Muestra cómo las entidades de nuestro modelo interactúan y cómo se programan los eventos (que han sido previamente seleccionados en el constructor).

El archivo Debug permite imprimir información para remover bugs del modelo. El archivo Error brinda un reporte de los errores cometidos en el manejo interno de DESMO-J (sacar elementos de colas a las que no pertenecen, mal manejo de estructuras, etc.).

Descripción del Reporte

Esta descripción ejemplo corresponde a la simulación del caso de Uso de Rivera, con 30 replicaciones, con las distribuciones desactivadas.

Cabe destacar que los datos que se muestran en el reporte corresponden a la última replicación de la serie de replicaciones, el único dato que corresponde a la serie de replicaciones es el histograma “Tiempo Medio de viaje de Pasajeros”.

En la Figura 75 se observa que el Reporte comienza con la descripción del modelo, y una descripción de la ejecución de la simulación, se pueden destacar los datos de tiempo de simulación (7200 segundos), y el tiempo computacional en el que lleva la ejecución es 3 segundos.

Description

Simulación transporte público FING

top

Simulation Run

Property	Content
Simulation duration	Experiment run from 0 until 7200.
Computation duration (HH:MM:SS)	00:00:03
Resets	No resets during the experiment run.
Seed	1008
Errors	No errors or warnings have occurred.

Figura 75 - Reporte de salida del simulador

Seguido por el reporte de histogramas utilizados, para los cuales se registran la cantidad de observaciones, el valor promedio, la desviación estándar, el valor mínimo y máximo registrado. Ver Figura 76 .

Tallies

Title	(Re)set	Obs	Mean	Std.Dv	Min	Max	Unit
Tiempo de viaje de Pasajeros en Replicación	0	882	2716.95238	1002.16502	561.0	6902.0	[Segundos]
Tiempo Medio de viaje de Pasajeros	0	30	2730.23985	26.43252	2672.66667	2783.88183	[Segundos]

Figura 76 - Histogramas del reporte del simulador

El histograma “Tiempo de viaje de Pasajeros en Replicación” muestra el resultado para la última replica-

ción, en cambio el histograma “Tiempo Medio de viaje de Pasajeros” muestra el resultado acumulado para todas las replicaciones.

El tiempo de viaje promedio de los pasajeros es de 2730 segundos (45 minutos), además se puede observar que la variación estándar es de 26 segundos (aproximadamente medio minuto).

A continuación, en la Figura 77 se muestran datos estadísticos de todas las colas definidas en el modelo, para cada una de ellas se puede observar datos de interés como cantidad de observaciones, cantidad (máxima / promedio / fin de ejecución) de entidades que se encolaron, tiempos de espera (máxima / promedio) en cola.

Queues

Title	Qorder	(Re)set	Obs	QLimit	Qmax	Qnow	Qavg.	Zeros	max.Wait	avg.Wait	refus.
cola de omnibus de parada: 142	FIFO	0	19	unlimit.	2	0	0.15833	0	60	60	0
cola de pasajeros de parada: 142	FIFO	0	39	unlimit.	11	1	3.55861	1	2000	603	0

Figura 77 - Colas del reporte del simulador

Por ejemplo, se puede observar que en la parada 142 se detuvieron 19 ómnibus ya sea para que asciendan o desciendan pasajeros. El promedio de espera en parada es de 1 minuto, lo que coincide con el valor fijado en la configuración. Otro dato que se puede observar, es que en la parada 142, la máxima cantidad de ómnibus que se encuentran en un mismo instante son 2.

Respecto a la cola de pasajeros de la parada, se puede observar que en promedio hay 4 pasajeros esperando para tomar el/los ómnibus especificados por la estrategia del pasajero, el pasajero que más esperó en la parada lo hizo por 33 minutos (2000 segundos).

Finalmente, se reportan todas las distribuciones utilizadas y sus valores de interés. Ver Figura 78 .

Distributions

Title	(Re)set	Obs	Type	Parameter 1	Parameter 2	Parameter 3	Seed
Velocidad de viaje de los buses	0	86	Cont Normal	166.6666717529297	33.33333206176758		7374602
Velocidad de Caminata de los pasajeros	0	1524	Cont Normal	50.0	10.0		9609295
Tiempo de Arribo de Omnibus	0	8	Cont Exponential	12.222249031066895			71529105

Figura 78 - Distribuciones del reporte del simulador

Anexo 6 - Manual de usuario

En este anexo se brinda el manual de usuario del conjunto de scripts que componen el proceso de transformación de datos de entrada. Primero se dan algunas nociones básicas sobre gvSIG y qué es necesario para la ejecución de los scripts. Luego se detalla en forma individual el procedimiento de ejecución para cada uno.

Los scripts están escritos en el lenguaje Jython y se ejecutan en el módulo Scripting Composer de gvSIG. La versión utilizada y donde están validados es gvSIG Desktop 2.2.0.2313 para Windows 64 bits.

gvSIG tiene un Gestor de Proyecto: es una ventana que permite gestionar tipos de documentos. Hay cuatro tipo de documentos: Vistas, Tablas, Mapas y Gráficas. En este anexo solo vamos a hablar de Vistas.

Para todos los scripts aplica que necesitan tener una Vista creada en el Gestor de Proyecto para su ejecución. También es común a todos, que en caso de ejecución exitosa se lanza un mensaje correspondiente. Para la ejecución de los scripts, primero se ejecuta gvSIG Desktop. Para abrir el Gestor de Proyecto, en caso que que no esté abierto, se debe ir al menú “Mostar/Gestor de Proyecto”. Accionando el botón “Nuevo” para crear una nueva Vista,como se ve en la Figura 79 .

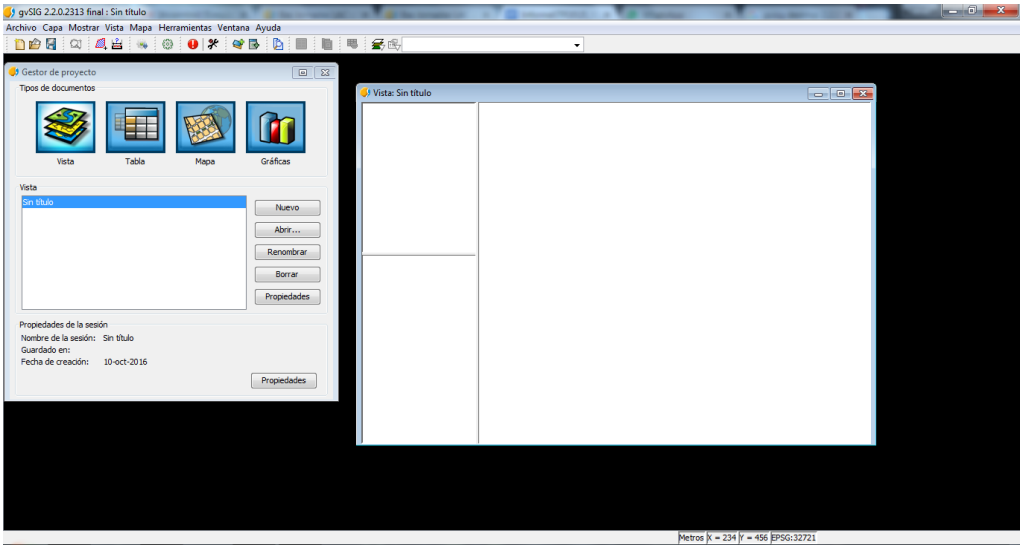


Figura 79 - gvSIG con Vista creada

En ese punto ir a “Herramientas/Scripting/Scripting Composer”. Se abrirá la ventana de Scripting Composer: es un explorador de scripts junto con un editor. La Figura 80 muestra el Scripting Composer abierto.

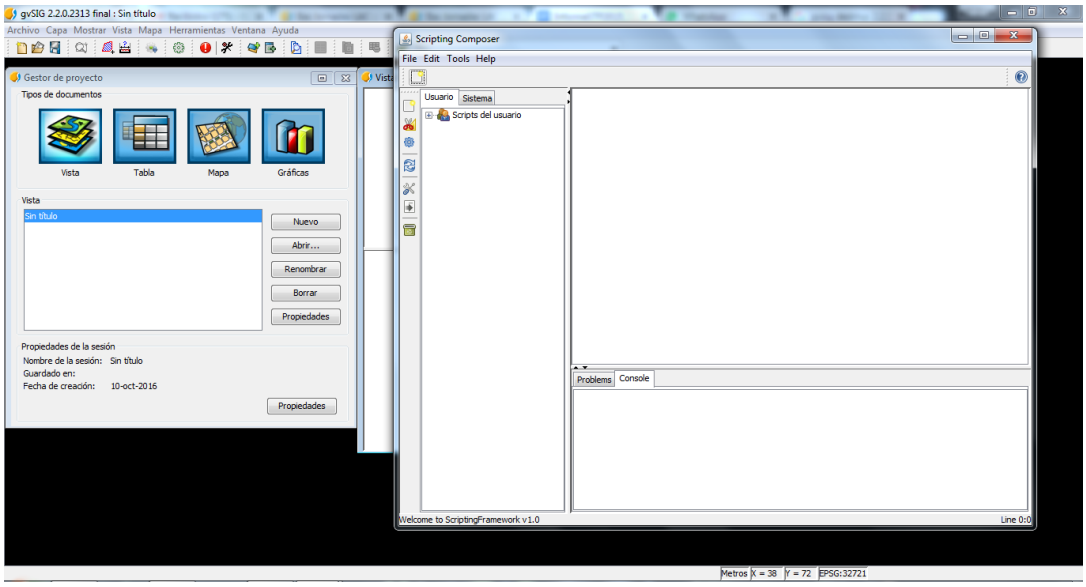


Figura 80 - Scipting Composer

Para instalar un script en Scripting Composer, seleccionar la opción “New” y luego copiar el código correspondiente al script.

unificarLineas.py

Con una Vista creada, ejecutar el script con el botón de “Run”. Se pide seleccionar una carpeta donde se almacenan los archivos de las líneas a unificar y dar “Aceptar”. En la Vista se carga cada archivo de línea tomado en cuenta en la ejecución, además de una nueva capa creada con las líneas unificadas. El nuevo shapefile creado, resultado de la ejecución se guarda en la carpeta seleccionada inicialmente.

crearIdParada.py

Para la ejecución de este script se debe tener una Vista creada, y una capa cargada y activa en la Vista. Para cargar un capa, con una Vista creada y abierta, con botón derecho del mouse seleccionar en la división rectangular superior izquierda (TOC de aquí en mas) y seleccionar del menú desplegado “Añadir capa”. Luego, seleccionar la capa de paradas a la que se le desea agregar un identificador y se cargará en la Vista como muestra la Figura 81 .

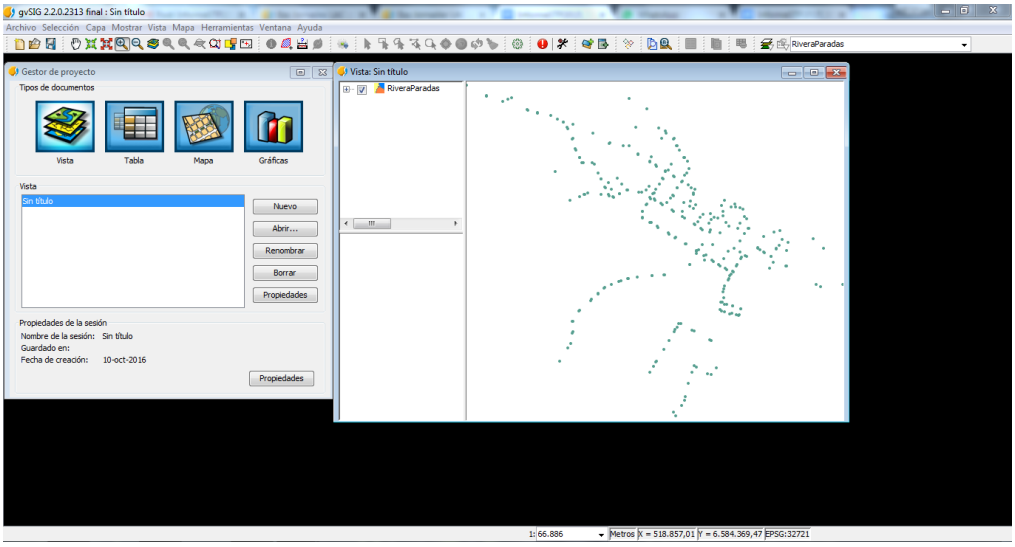


Figura 81 - Capa cargada

Para activar la capa seleccionar sobre el nombre de la capa que se cargó en el TOC. Una vez activa el nombre quedará en negrita. Luego ejecutar el script con el botón de “Run”.

crearIdZona.py
Este script se ejecuta de forma análoga al script anterior.

setParadaLineaSeleccion.py
Para la ejecución de este script es necesario tener la capa de paradas y la de lineas a modificar cargadas en una Vista, antes de la ejecución se debe tener la parada seleccionada que se desea agregar a una línea y conocer el identificador de esta última.
Ejecutar el script con el botón de “Run”. A continuación se pide el ingreso del nombre de la capa de líneas a modificar. Luego se pide el ingreso del nombre de la capa de paradas de donde tomar las paradas. A continuación se pide el ingreso del identificador de la línea a la cual se le asignará la parada. Concluida la ejecución, la línea en cuestión tendrá asociada la parada a nivel de atributos de la capa. Por cada parada a asignar se debe realizar un ejecución individual.

agregarParadasLineaFile.py
Para la ejecución de este script se debe tener la capa de líneas cagada en el TOC. Ejecutar el script con el botón de “Run”. A continuación se pide el nombre de la capa de líneas con que trabajar, luego se pide el ingreso del archivo con que configurar las paradas.

Anexo 7 - Manual del desarrollador

En este Anexo se explica cómo es la estructura del desarrollo de la aplicación y también cómo poder agregar, activar y desactivar estrategias para las pruebas del sistema. Luego se detalla la manera de generar un ejecutable de la aplicación creado desde Eclipse.

Estructura de la aplicación

El aplicación es realizada en el lenguaje Java, tanto las bibliotecas utilizadas y los desarrollos implementados. Las bibliotecas utilizadas son:

- para el simulador: DESMO-J v2.4.2 (Noviembre 2014)
- para el SIG: GeoTools v14.M0 del (Mayo 2015)
- para entorno visual: SWT para Windows 64 bits.

El entorno de ejecución se configuró en JavaSE-1.7 y se utilizó el software Eclipse como IDE para el desarrollo.

La estructura del desarrollo consta de dos componentes que separan el simulador del visualizador como se muestra en la Figura 82 .

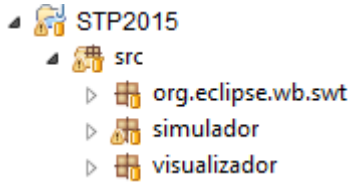


Figura 82 - Estructura General

A su vez el simulador se divide en la estructura base del simulador y la implementación del simulador orientado a eventos discretos. El visualizador se divide en interfaz gráfica e implementación del SIG. La Figura 83 muestra lo antes descrito.

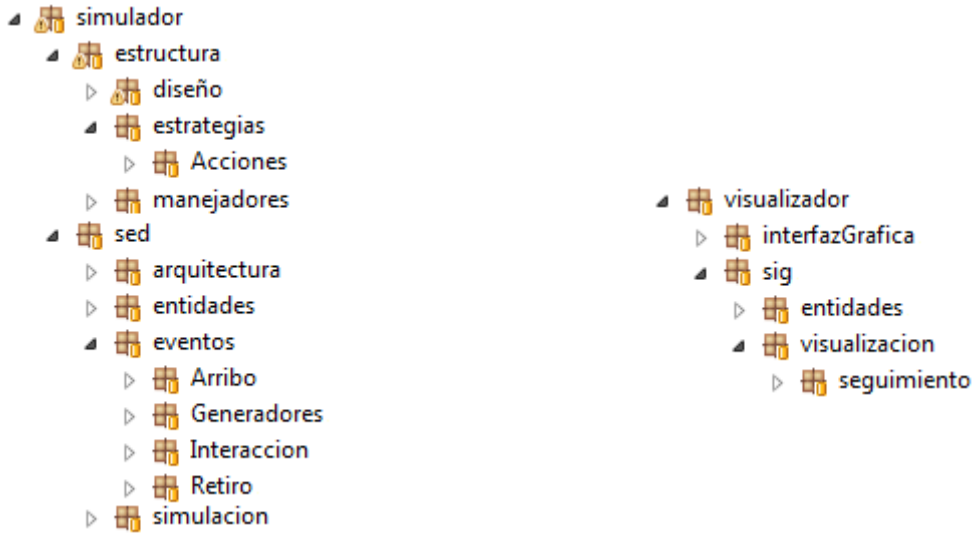


Figura 83 - Estructuras del simulador y del visualizador

Un punto importante en el desarrollo, es la creación de estrategias para realizar estudios sobre diferentes comportamientos de los pasajeros. Para poder habilitar una estrategia en el sistema -además de implementar su funcionalidad-, debe ser creada la clase correspondiente en la ruta "*src/simulador/estructura/estrate-*

gias". Luego en el enumerado "*estrategiasPublicadas*" -que se encuentra en la misma ruta- se debe registrar el nombre de la estrategia creada. De esta manera el sistema detecta qué clases debe usar para asignar a los usuarios las diferentes estrategias.

Crear ejecutable de la aplicación

A continuación se detalla la creación -desde Eclipse- de un archivo ejecutable para la aplicación.

En primer lugar se debe tener creado un lanzador de ejecución el cual servirá para informar al archivo final cuál es la clase inicial de la aplicación. Para esto, -desde eclipse y con el proyecto abierto- se selecciona la opción "Run/RunConfigurations...", como se ve en la Figura 84 .

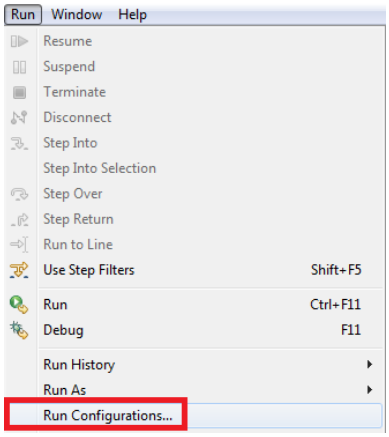


Figura 84 - Crear lanzador (1)

Luego aparecerá un cuadro de diálogo para configurar el lanzador. Allí se debe seleccionar la opción "Java Applicaction", y apareceran los campos correspondientes a la configuración de esta opción. Allí se deben completar los campos "Name" -este es el identificador del lanzador, puede tener cualquier nombre-, "Proyect" -es el nombre del proyecto, en este caso STP2015- y "Main Class" -es la clase inicial de la aplicación, en este caso es visualizador.interfazGrafica.Principal. Luego de esto, se selecciona el botón "Apply" para guardar la configuración del lanzador. En la Figura 85 se ve la configuración descrita.

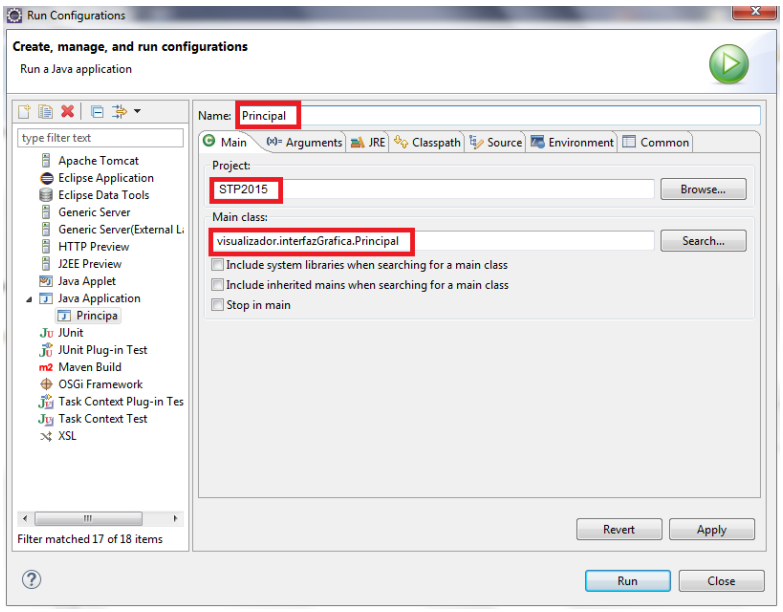


Figura 85 - Crear lanzador (2)

Una vez creado el lanzado, se apreta con el botón derecho del mouse sobre la raíz del proyecto y se selec-ciona la opción “export..”, como se muestra en la Figura 86 .

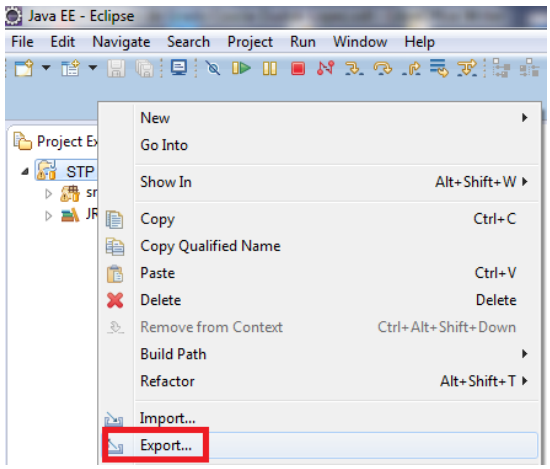


Figura 86 - Crear ejecutable (1)

Una vez seleccionada esta opción, aparecerá un cuadro de diálogo para elegir la forma de exportar el pro-yecto. En este caso se debe elegir la opción “Java/Runnable JAR file” y se presiona el botón “Next”, tal como se ve en la Figura 87 .

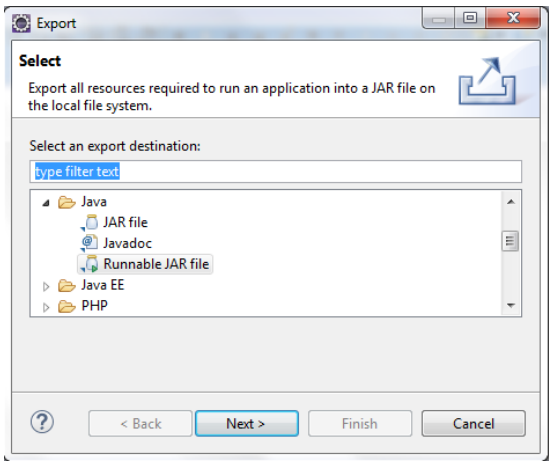


Figura 87 - Crear ejecutable (2)

Aparecerá la cuadro de diálogo para la configuración del ejecutable. Allí se debe seleccionar el lanzador configurado previamente en “Launch configuration:”, una dirección de destino del archivo ejecutable a crear y se debe seleccionar la opción “Package required libraries into generated JAR”. Luego se presiona el botón “Finish” y se crea el archivo ejecutable en la dirección especificada. En la Figura 88 se ve un ejemplo de la configuración.

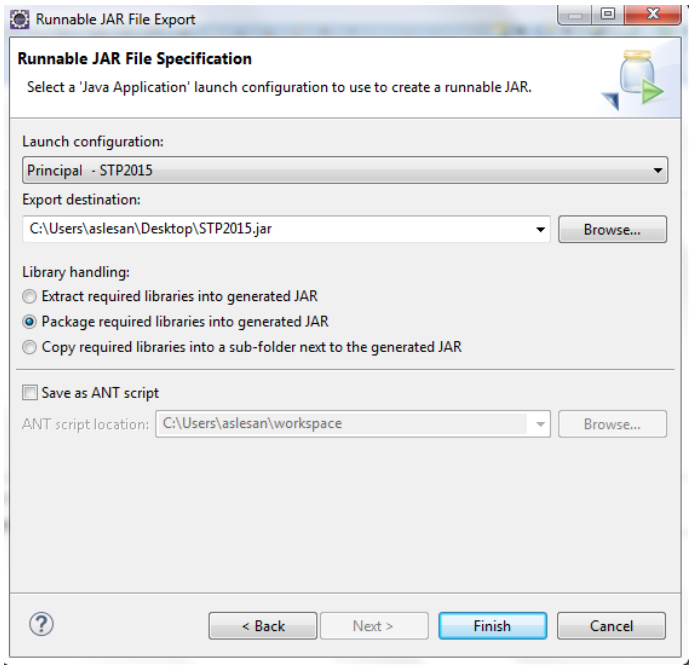


Figura 88 - Crear ejecutable (3)

Por último para que se pueda ejecutar el archivo creado, las imágenes que se encuentran en el proyecto deben estar en el mismo directorio donde se encuentra el ejecutable, bajo la carpeta “images”. En la Figura 89 se muestra como debe quedar el ejecutable con la carpeta de las imágenes.

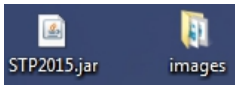


Figura 89 - Archivo ejecutable creado

Bibliografía

[1] M. Estrada, E. Nacclle, L. Segura; Simulación de sistemas de transporte público con servicios de información a usuarios en tiempo real; Proyecto de Grado de Ingeniería en Computación, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay; 2013

[2] D. Gawenda, H. Martínez; Interfaz para herramienta deplanificación de recorridos paratransporte público; Proyecto de Grado de Ingeniería en Computación, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay; 2008

[3] P. Banchero, R. Giesen, A. Mauttone; Effect of passenger behavior and service characteristics over the performance of bus transit systems, a simulation study; Ninth Triennial Symposium on Transportation Analysis (TRISTAN IX), Oranjestad, Aruba; 2016

[4] M. Estrada, R. Giesen, A. Mauttone, E. Nacelle, L. Segura; Experimental evaluation of real-time information services in transit systems from the perspective of users; Conference on Advanced Systems in Public Transport (CASPT 2015), Rotterdam, Holanda; 2015

[5] Avishai (Avi) Ceder; Public Transit Planning and Operation; Civil and Environmental Faculty, Transportation Research Institute,Technion-Israel Institute of Technology, Haifa, Israel; 2007

[6] M. López, P. Lorenzo, P. Medina; Herramientas para la simulación del transporte público urbano colectivo; Proyecto de Grado de Ingeniería en Computación, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay; 2011

[7] P. Aldaz, G. De León; Simulador de transporte público urbano colectivo; Proyecto de Grado de Ingeniería en Computación, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay; 2009

[8] R. Davies, R. O'Keefe; Simulation Modelling whit Pascal; Prentice Hall International Ltd.; 1989

[9] P. Rebufello; Introducción a los SIG; https://www.fing.edu.uy/inco/cursos/sig; 2012

[10] QGIS; Datos Vectoriales; julio 2015

[11] ESRI; http://www.esri.com; julio 2015

[12] Sistema de coordenadas de referencia; https://docs.qgis.org/2.8/es/docs/gentle_gis_introduction/coordinate_reference_systems.html; julio 2015

[13] E. Horowitz, S. Sahan, S. Rajasekaran; Computer Algorithms, 2nd Edition, ; Computer Science Press; 1997

[14] QGIS; http://www.qgis.org/es/site/; julio 2015

[15] gvSIG; http://www.gvsig.com/; marzo 2016

[16] ArcGIS; https://www.arcgis.com/features/index.html; julio 2015

[17] GeoTools; http://www.geotools.org/; marzo 2016

[18] EOSimulator; https://www.fing.edu.uy/inco/cursos/simulacion/eosim_html/index.html; julio 2015

[19] DESMO-J; http://desmoj.sourceforge.net/home.html; marzo 2016

[20] AGESIC; https://catalogodatos.gub.uy/; marzo 2016

[21] INE; http://www.ine.gub.uy/; marzo 2016

[22] J. Ortúzar, L. Willumsen; Modelling Transport, 4th Edition; Wiley; 2011

[23] SAGA; http://www.saga-gis.org/en/index.html; diciembre 2015