Instituto de Computación – Facultad de Ingeniería
Universidad de la República

Tesis de Maestría
en Ingeniería en Computación

Transporte Vertical: Simulación y Análisis de Sistemas de Ascensores

Andrés Hetzel

Director: Dr. Héctor Cancela

Montevideo, Uruguay
Abril de 2003

Transporte Vertical: Simulación y Análisis
de Sistemas de Ascensores
Andrés Hetzel

ISSN 1510-7264
Tesis de Maestría en Ingeniería en Computación
Instituto de Computación – Facultad de Ingeniería
Universidad de la República

Montevideo, Uruguay, Abril de 2002

RESUMEN

En este trabajo analizamos la historia y el estado del arte de los sistemas de control de grupos de ascensores, y la simulación como herramienta para su desarrollo. Pasamos revista a las investigaciones matemáticas sobre el problema, concluyendo que es un problema NP-Completo. Luego analizamos y clasificamos los distintos tipos de algoritmos y estrategias diseñados para atacarlo. Finalmente implementamos un simulador capaz de utilizar dichos algoritmos y lo utilizamos para estudiar el impacto de los fenómenos de Balking y Reneging en estos sistemas.

PALABRAS CLAVE: ascensores, simulación, ruteo, control de grupos.

ABSTRACT

In this paper we analyze the history and state of the art of elevator group control systems, and the use of simulation as a tool for their developement. We review the mathematical research done on this problem, and reach the conclusion that it is a NP-Complete problem. After that, we analyze and classify the different kinds of algorithms and strategies designed to approach it. Finally, we implement a simulator capable of using those algorithms and we use it to study the impact that the phenomena known as Balking and reneging have on these systems.

KEYWORDS: elevators, simulation, routing, group control.

INDICE

1) INTRODUCCIÓN

El objetivo de esta tesis de maestría es la investigación de las aplicaciones de la Investigación Operativa al problema del control de los sistemas de ascensores.

Los ascensores tienen una larga historia, y lo mismo vale para sus sistemas de control. Comenzamos entonces en el capítulo 2 con un breve recuento del desarrollo de la tecnología utilizada para el control y ruteo de los ascensores, desde los ascensoristas humanos hasta la incorporación de sistemas informáticos.

Si consideramos el control de un ascensor o de un grupo de ascensores desde un punto de vista matemático, queda claro es un problema de ruteo de vehículos sobre una red de caminos simplificada (pues es lineal). Investigando la literatura, encontramos que se han definido muchas variantes de problemas que modelan con mayor o menor grado de aproximación la realidad de un único ascensor. Constatamos que está demostrado que estos problemas son NP-completos, y que se han desarrollado algoritmos polinomiales que arrojan soluciones aproximativas acotadas. Sin embargo, no encontramos trabajos equivalentes para el ruteo de varios ascensores. Por lo tanto decidimos definir un problema que de forma similar constituya un modelo aproximado del caso de más de un ascensor, y demostramos que también es NP-completo. Los detalles sobre nuestra investigación se encuentran en el capítulo 3.

Si bien para este problema no hay soluciones obtenibles en tiempo polinomial cuyo grado de eficacia esté matemáticamente demostrado, sí existen algoritmos y estrategias que brindan soluciones aceptables. Esto queda evidenciado por la existencia de sistemas comerciales de control de ascensores. Desgraciadamente, las compañias de ascensores tienen tendencia a mantener los detalles de sus sistemas secretos y solo mencionan generalidades. No obstante, hay suficientes trabajos sobre ruteo de vehículos aplicados a ascensores como para recopilar un panorama general del estado del arte de la informática respecto a este problema. Encontramos que los distintos algoritmos y estrategias existentes se pueden clasificar en siete grandes grupos. De esto trata el capítulo 4.

Estudiando la evolución del ruteo de ascensores surge que una de las herramientas más valiosas para el análisis y desarrollo de algoritmos de control de ascensores son los simuladores. Sin embargo, hay una falta de programas de simulación de ascensores genéricos, debiendo cada investigador desarrollar el suyo propio. Decidimos intentar paliar esta duplicación de esfuerzo creando un simulador de sistemas de ascensores que fuera lo más versátil posible. En el capítulo 5 describimos el programa resultante, el SSA, especificando su estructura, sus entradas y sus salidas. En el capítulo 6, validamos nuestro sistema contra los resultados obtenidos con otro simulador por Cassandras et al [15], y lo utilizamos para analizar la relevancia del Balking y Reneging, dos fenómenos rara vez tenidos en cuenta en el diseño de este tipo de simulaciones.

Finalmente, en el capítulo 7 recopilamos las conclusiones a las que llegamos durante este trabajo. Completamos este informe con el detalle de la bibliografía que utilizamos, y cuatro apéndices: (A) el manual de usuario para el SSA, (B) el procedimiento para definir nuevos algoritmos de ruteo en el SSA, (C) la especificación del formato de entrada de datos del SSA y (D) la especificación de su formato de salida.

2)BREVE HISTORIA DE LA TECNOLOGÍA DE ASCENSORES

El concepto del ascensor es uno muy antiguo en la historia humana, pero ha sido apenas en los últimos 150 años que tomó forma el concepto del ascensor de pasajeros modernos. El punto crucial fue la invención del freno de seguridad automático por Elisha Otis en 1852. Esto permitió transformar a los ya existentes sistemas de montacargas hidráulicos o de vapor en vehículos viables para el transporte masivo de pasajeros. De allí en más, la tecnología del transporte vertical ha sido mejorada permanentemente, impulsada por la construcción de edificios cada vez más altos, y viceversa [1].

Es posible clasificar dichas mejoras en dos áreas distintas. En primer lugar están los sistemas mecánicos del ascensor, principalmente los motores. Estos han cambiado del uso del vapor a la electricidad, pasando de sistemas de pistones a dispositivos de tracción con o sin engranajes, y recientemente motores lineales sin cuerdas [1][2][3]. En segundo lugar, están los sistemas de control del ascensor, que son nuestro principal foco de interés aquí.

Los primeros ascensores usaban dispositivos de control mecánicos simples, tales como el control de cuerda, en el que el pasajero debía tirar de una cuerda para hacer que el ascensor se moviera en la dirección deseada. Algo más tarde se desarrollaron palancas eléctricas para el control del movimiento, y apareció la figura del ascensorista para encargarse de su manejo, lo que aumentaba la seguridad y confiabilidad del sistema.

En los años veinte aparecieron los sistemas de control por botoneras, al principio para la comodidad del usuario para llamar al ascensorista, y luego para el control del destino del ascensor. Poco después se diseñaron los primeros sistemas semiautomáticos, capaces de enviar las llamadas directamente al ascensor si no hay ningún pedido en curso.

En los años cincuenta, fueron inventadas las puertas automáticas para los ascensores. Esto, combinado con la tecnología de relés eléctricos desarrollada sobre todo por las compañías de telefonía, permitió diseñar sistemas de ascensores totalmente automáticos, prescindiendo de ascensoristas. Es entonces que la complejidad de los sistemas de ruteo empezó a aumentar, cambiándose por ejemplo el pulsador único de llamada por dos, uno para pedir la subida y otro para la bajada [2]. Esto daba más información al sistema de control, mejorando la eficiencia. Es también en esta época que las estrategias para el control de múltiples ascensores se diferenciaron de las usadas para controlar un único ascensor. Se desarrolló el concepto de control de grupos, en el que los pedidos de parada son enviados a un sistema de control central, el cual elige a qué ascensor asignárselo. De esta forma, se pasó de estrategias individuales que al trabajar en paralelo en cada ascensor brindaban un resultado general aceptable a un único algoritmo central capaz tanto de mejor eficiencia como de manejar situaciones excepcionales.

En los años setenta se empezó a aplicar la electrónica al control de ascensores [2]. En un principio las estrategias desarrolladas para los sistemas de relés fueron directamente replicadas, pero las mayores posibilidades de los circuitos electrónicos no tardaron en ser aprovechadas. Asignación de prioridades a las llamadas más viejas, estrategias variables según la demanda, y otras mejoras a la eficiencia se volvieron posibles. Se estableció el concepto de zonas dentro de los edificios, conjuntos de pisos atendidos solo por un subgrupo de ascensores. También se comenzó a usar la

simulación para probar los diseños de los sistemas (los primeros simuladores eran modelos analógico-digitales de generación de tráfico conectados a los auténticos sistemas de control).

Con la aparición del microprocesador en los años ochenta, los sistemas de control de los ascensores entraron en el ámbito de la informática, y su complejidad se ha disparado desde entonces. Uno de los primeros avances, por ejemplo, fue la priorización de las llamadas no sólo por su antigüedad real sino por la predicha por algoritmos estadísticos. Otro recurso que fue mejorado fue el de la simulación. Con simuladores puramente de software ejecutándose en minicomputadores, se pudieron verificar y optimizar varios algoritmos de ruteo y de identificación de condiciones de tráfico, además de impulsar el desarrollo de nuevas formulas para su análisis estadístico [3].

El permanente aumento en velocidad y disponibilidad de los microprocesadores eventualmente llevó la inteligencia artificial a los sistemas de contro l de ascensores. Muchos tipos de sistemas expertos fueron diseñados y desplegados por diferentes compañías, previstos para optimizar diversos aspectos del sistema (algunos mejoraban tiempos de viaje, otros se concentraban en las situaciones de mayor demanda). Más adelante se los combinó con sistemas de lógica difusa para enfrentar mejor la inherente incertidumbre de las premisas en que los expertos se basaban para definir sus reglas [3][4].

Recientemente se han desplegado los primeros sistemas de ascensores cuyo control incorpora redes neurales. Este tipo de controlador tiene la ventaja de poder aprender y optimizarse a sí mismo a medida que pasa el tiempo, en efecto personalizándose para el edificio en que está instalado. Se han diseñado controladores en los que la red neural efectúa tareas diversas: en algunos implementa la lógica difusa de sistemas ya existentes, en otros se la usa para identificar los patrones de tráfico a fin de elegir el algoritmo clásico más ajustado a la situación, y en unos pocos se la usa como control central para hacer la asignación de llamadas con el objetivo de minimizar las esperas [3].

En el futuro se puede esperar que los sistemas de control se vuelvan aún más inteligentes, incorporando los estudios que se realizan en varios campos, como reconocimiento visual para la identificación exacta de los flujos de tráfico, o los algoritmos genéticos para el control.

3) CONSIDERACIONES MATEMÁTICAS

3.1) Consideraciones Preliminares

En este capítulo pasaremos revista a los estudios realizados sobre sistemas de ascensores desde un punto de vista matemático y estadístico. Hemos encontrado en la literatura dos áreas distintas que responden a esta definición. Por un lado, el análisis matemático de cómo optimizar el ruteo de un ascensor, y por el otro, el análisis estadístico de las capacidades de un sistema de ascensores necesarias para satisfacer diferentes tipos y volúmenes de demanda.

Empezaremos definiendo cuáles son los problemas de ruteo asociados a los ascensores y qué estudios se han realizado sobre ellos. Luego observaremos que todos esos problemas se centran en el ruteo de un único ascensor, así que definiremos un nuevo problema que esté asociado al ruteo de varios ascensores a la vez. Finalmente, revisaremos las conclusiones más relevantes de los estudios estadísticos realizados sobre los sistemas de ascensores, con especial énfasis en los estándares internacionales de diseño.

Cabe notar que para todos lo casos mencionados, un concepto de base es cómo medir el comportamiento de los sistemas de ascensores, requisito indispensable para poder determinar si un sistema se comporta mejor que otro.
El parámetro más común es el tiempo de espera de un pasajero desde que llama al ascensor hasta que éste llega. En los sistemas de control se intenta reducir el tiempo de espera medio, el tiempo de espera máximo, el porcentaje de pasajeros que esperan más que un tiempo dado (p.ej, 60 segundos), o la suma de los cuadrados de los tiempos de espera (para estimular una reducción más uniforme que con el promedio). El otro parámetro común es el tiempo total que el pasajero pasa en el sistema, y se lo maneja de manera similar al tiempo de espera, minimizando sus promedios, máximos y cuadrados. Existen otros parámetros secundarios, como consumo de energía, número de arranques y paradas, o coeficientes de satisfacción del pasajero; estos son usados sobre todo por sistemas de control más sofisticados, que intentan balancear la optimización simultánea de muchos parámetros [2][5][4].

Como veremos, el análisis estadístico de los sistemas de ascensores ha utilizado tanto el tiempo de espera de los pasajeros como el tiempo que pasan en el sistema para calificar la performance.
Los análisis matemáticos del ruteo de ascensores, por su parte, se efectúan sobre simplificaciones basadas en grafos y calculan costos a partir de los pesos de las aristas. Esto les da un cierto nivel de abstracción, en la medida que los pesos pueden representar tanto distancias como tiempos o consumo de energía, pero implica que más que optimizar la experiencia individual del pasajero (que es el objetivo principal en sistemas comerciales), optimizan las propiedades de la trayectoria del ascensor.

3.2) Análisis del Problema del Ruteo de un Ascensor

3.2.1) Definición del Problema

El problema del ruteo de un ascensor puede ser considerado como una variante de uno de los problemas clásicos de la optimización combinatoria: el problema del vendedor viajero, o TSP (Travelling Salesman Problem). Es generalmente desde ese punto de vista que ha sido analizado en la literatura. Sin embargo, presenta características únicas que lo convierten en un caso muy particular.
Hay dos problemas clásicos derivados del vendedor viajero que se aproximan más a la descripción de un ascensor [6][7][9]. Estos son :

DARP (Dial-A-Ride Problem): Este escenario consiste de un sistema de transporte, una red de caminos (grafo ponderado) y una serie de n objetos que deben ser transportados de un punto del camino a otro. El transporte sólo puede llevar un objeto por vez. El objetivo es encontrar el camino más corto para mover todos los objetos de sus respectivos orígenes a sus respectivos destinos.
Hay diferentes versiones de este problema:

- Si el transporte puede dejar un objeto que transporta en un punto de espera para mover otros objetos antes de llevarlo a su destino final, se le llama **preemptivo**. Si por el contrario cuando el transporte recoge un objeto solo lo puede dejar en su destino final, se le llama **no preemptivo**.
- Cuando efectuar todas las entregas el transporte debe volver a su punto de origen, se le llama **cerrado**. Si no tiene esta exigencia, se le dice **abierto**.

CDARP (Capacitated Dial-A-Ride Problem): Es idéntico al DARP, pero el transporte puede llevar un numero dado k de objetos al mismo tiempo (a todos los efectos, DARP es CDARP con capacidad uno)

En [9] está planteada una definición formal del CDARP, que reproducimos aquí como referencia:

> Sea un grafo mixto G = (V,E,R) un conjunto V de vértices, un conjunto E de aristas sin paralelos y un multiconjunto R de arcos dirigidos (pueden ser paralelos).
> Una instancia de CDARP es un grafo mixto finito G=(V,E,R), cuyas aristas tiene pesos dados por la función d: $E \to \mathbf{R}_{\geq 0}$, un transporte de capacidad $C \in \mathbf{N}$ y una posición de inicio del transporte $o \in V$. La red de transporte está representada por E y los objetos a transportar están representados por los arcos R. Cada objeto $r \in R$ debe ser llevado por el transporte del vértice origen $\alpha(r) \in V$ al vértice destino $\omega(r) \in V$.

El movimiento del transporte del vértice v al w llevando un conjunto de objetos Q se representa como (v,w,Q). (La cantidad |Q| de objetos en un movimiento debe ser menor o igual que la capacidad C)
Una transportación es una serie de movimientos $T = (v_1,v_2,Q_1)\ (v_2,v_3,Q_2)\ ...\ (v_h,v_{h+1},Q_h)$.
El costo de la transportación T es $C(T) = \sum_{i=1}^{h} d(v_i, v_{i+1})$ donde $d(v_i, v_{i+1})$ es la menor distancia entre v_i y v_{i+1} en G. Un objeto r es movido por T de v_1 a v_{h+1} si $r \in Q_i$ para $i = 1..h$.
Una transportación factible no preemptiva para una instancia de CDARP es una transportación $T = (v_1,v_2,Q_1)\ (v_2,v_3,Q_2)\ ...\ (v_h,v_{h+1},Q_h)$ tal que cumple:

- T empieza y termina en el vértice o (o sea, $v_1 = v_{h+1} = o$)
- Para todo objeto $r \subset R_i$, la subsecuencia de movimientos (v_j,v_{j+1},Q_j) de T que cumplen $r \subset Q_j$ es una subsecuencia contigua de T y forma una transportación de $\alpha(r)$ a $\omega(r)$.

El objetivo del problema CDARP es encontrar una transportación factible de costo mínimo.

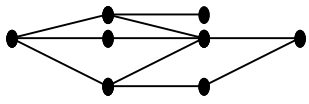
Otras dos versiones del DARP han sido definidas por Sven Krumpke [6] para aproximar algunos aspectos del problema a la realidad de un ascensor. Estas son:

SOURCE-DARP: Es idéntico al DARP, pero se le agrega a cada conjunto de objetos que comparten el mismo punto de origen un orden de entrega que debe ser respetado. Es decir, si A y B tienen el mismo origen y el orden de A es menor que el de B, A debe ser entregado primero. Este orden no acarrea precedencias entre objetos con distintos orígenes. (DARP es SOURCE-DARP donde todos los objetos tienen igual orden 0).

PENALTY-SOURCE-DARP: Es idéntico al SOURCE-DARP, pero cada vértice del camino tiene asociado un costo de parada y un costo de arranque que se suman al costo total de la solución. (SOURCE-DARP es PENALTY-SOURCE-DARP con valores de arranque y parada 0 en cada vértice).

Como hemos podido ver, las descripciones de estos problemas tienen una gran semejanza con la de un único ascensor. Debemos puntualizar de todos modos dos diferencias significativas:
Primero, todas estas versiones del DARP y CDARP definen la red de caminos como un grafo cualquiera, mientras que en el caso del ascensor, el camino a ser recorrido es un grafo lineal, lo que constituye un caso particular de esos problemas.

Caso del ascensor



Caso CDARP Genérico

Segundo, en estos problemas se presupone que todos los objetos, sus orígenes y destinos son conocidos de antemano, y se debe minimizar el total del tiempo; en el caso real de un ascensor, este conocimiento simplemente no esta disponible, y el minimizar el tiempo total no necesariamente seria un buen criterio para optimizar el transporte de personas.
A pesar de dichas diferencias, el análisis de DARP, CDARP y sus variantes es relevante para el problema del ascensor; inclusive, como veremos, estas han sido tomadas en cuenta en algunos trabajos.

3.2.2) Estudios y sus Conclusiones

El primer trabajo que debemos mencionar es el de D. J. Guam en 1996 [7]. En el mismo fueron demostradas ciertas importantes propiedades del CDARP para capacidades mayores o iguales a dos, a saber:

- CDARP no preemptivo en un grafo lineal es NP-completo.
- CDARP preemptivo en un grafo lineal es resoluble en tiempo polinomial.
- CDARP en un grafo tipo árbol es siempre NP-completo.
- también se menciona que era conocido que DARP siempre es polinomial en un grafo lineal, pero NP-completo en un grafo tipo árbol.

Estos resultados demuestran que no es factible el uso de un algoritmo para buscar la solución optima al ruteo de un ascensor en sistemas de gran tamaño. Por esto las investigaciones en este campo se han orientado a la búsqueda de algoritmos aproximativos en tiempo polinomial que brinden una solución de competitividad acotada (se dice que un algoritmo es j -competitivo si en el peor de los casos da una solución de costo j veces el de la solución optima [6]).

Algunos trabajos interesantes en esa área son los de Charikar y Raghavachari [8] y Krumke et al [9]. En el primero se plantea un algoritmo para el CDARP con capacidad k en un grafo genérico de competitividad $O(\log n \log\log n)$ si es preemptivo, y $O(\sqrt{k} \log n \log\log n)$ si no lo es (donde n es el número de vértices del grafo). También anuncia como teorema la existencia de un algoritmo de competitividad 2 para el CDARP en un grafo lineal, pero no ha publicado la demostración ni el algoritmo.
En el segundo se plantea un algoritmo para CDARP no preemptivo en grafos lineales y se demuestra que es 3-competitivo.

Posteriormente, Krumpke [6] analizó los problemas SOURCE-DARP y PENALTY-SOURCE-DARP. Demostró que SOURCE-DARP es soluble en tiempo lineal para grafos lineales y NP -completo para otros grafos, mientras que PENALTY-SOURCE-DARP es siempre NP-COMPLETO. También planteó algoritmos de aproximación para ambos problemas, que son de competitividad 3/2 para grafos tipo árbol y 9/4 para grafos genéricos.

Como mencionamos antes, todos estos trabajos tratan de la situación en que se conoce desde el principio todos los objetos a transportar, lo que no refleja correctamente la realidad de un ascensor. Ascheuer, Krumpke y Rambau [10] definieron lo que llaman “Online Transportation Problem”, que fundamentalmente consiste en un DARP en que los objetos a transportar aparecen en secuencia, y

no se espera a que aparezcan todos antes de empezar a transportarlos. Luego analizaron como resolver el problema usando los llamados “algoritmos online”.
Un algoritmo online determinístico recibe como entrada una secuencia de pedidos, cada uno con un instante de llegada asociado, los que determinan el orden y el momento en que cada pedido de la secuencia es revelado al algoritmo. Su comportamiento en un instante t dependerá entonces de t y de los pedidos que hayan sido revelados hasta ese momento. La eficiencia de este tipo de algoritmos se llama *competitividad* y es medida comparando el costo de sus resultados con los de un algoritmo “offline” óptimo (es decir, conoce toda la secuencia desde el principio y devuelve la solución de menor costo posible). Se dice que un algoritmo online es de competitividad c (o c -competitivo) si para toda secuencia la solución que devuelve es a lo sumo c veces más costosa que la que devuelve el algoritmo o ffline óptimo [6].
Ascheuer, Krumpke y Rambau demostraron entonces que no existe ningún algoritmo online determinístico para resolver este problema de competitividad menor que 5/3 para la versión cerrada ($1+\sqrt{2}/2$ en grafos lineales) y 2-4/D (con D la máxima distancia entre un vértice y el vértice de origen del transporte) para la versión abierta.
Finalmente, plantearon un par de estrategias determinísticas simples (REPLAN, IGNORE) para resolver este problema, y demostraron que son de competitividad 5/2 para la versión cerrada y 3 y 4 respectivamente para la versión abierta.

Más adelante, Krumpke [6] extendió este análisis del problema online al CDARP. Menciona una demostración para la versión cerrada que establece una cota inferior de competitividad de 2 para algoritmos determinísticos en grafos generales y la extiende, encontrando que en la línea se sigue cumpliendo la cota $1+\sqrt{2}/2$ hallada para el DARP-online, mientras que para la versión abierta 2 es una cota inferior para todos los grafos.
Luego analiza los algoritmos REPLAN e IGNORE, y demuestra que para CDARP-online son de competitividad 7/2 y 5/2 para la versión cerrada, y 9/2 y 4 para la versión abierta. También plantea un nuevo algoritmo determinístico, SMARTSTART, de competitividad 2 (o sea óptimo) para la versión cerrada y $2+\sqrt{2}$ para la versión abierta.

No hay aun análisis similares para SOURCE-DARP-online o PENALTY-SOURCE-DARP-online, pero es razonable suponer que eventualmente se aplicarán algoritmos online a esos problemas, y se demostrarán cotas inferiores de competitividad para ellos. Un buen lugar para seguir la evolución de estos estudios es la pagina web de Sven Krumpke y Jörg Rambau [11].

3.3) Análisis del Problema del Control de Grupos de Ascensores

Curiosamente, no parece haber en la literatura ningún trabajo que analice este problema desde un punto de vista matemático, como los hay para el del ruteo de un ascensor.

Como contribución a este emprendimiento definiremos y analizaremos aquí una variante del problema del vendedor viajero que tenga una relación similar al control de grupos que la que tiene CDARP al ruteo de un ascensor.

MCDARP (Multiple-transport Capacitated Dial-A-Ride Problem):

Este problema consiste de *m* sistemas de transporte de capacidad máxima *k*, un grafo ponderado y una serie de *n* objetos que deben ser transportados de un vértice del grafo a otro. Los transportes se encuentran todos inicialmente en un mismo vértice de origen *o*. El objetivo es encontrar un conjunto de *m* caminos para el recorrido de los transportes tal que transporte todos los objetos a sus respectivos destinos finales y que minimice la suma de los cuadrados de los costos de cada camino. (Cuando *m*=1 , MCDARP y CDARP tienen la misma solución óptima)

Elegimos minimizar la suma de los cuadrados de los costos en vez de la de los costos porque es una manera simple (y usada en auténticos algoritmos de control de ascensores) de expresar nuestro deseo de balancear la carga entre los transportes y, si el costo representara el tiempo que consume un movimiento, minimizar el tiempo total de entrega de los objetos.
Originalmente consideramos usar simplemente la suma de los costos, pero resultó ser un criterio ineficaz, dado que es trivial demostrar (al menos para la versión cerrada) que se puede obtener una solución óptima usando uno solo de los *m* transportes, lo que efectivamente maximiza el tiempo total de entrega.
Otra alternativa que consideramos fue elegir la solución que minimice el costo del mayor camino. Esto impone una eficaz cota al tiempo total de entrega, pero es ambiguo respecto al balance de carga. Para usar este criterio haría falta una definición recursiva que minimice el costo del mayor de los *m* caminos, luego el mayor de los *m-1* caminos restantes, y así sucesivamente hasta llegar al final. Aunque tiene un cierto interés, consideramos que agrega demasiada complicación para obtener un resultado equivalente al de la suma de cuadrados.

Basándonos en la definición de CDARP planteada más arriba, podemos definir formalmente MCDARP de la siguiente manera:

Sea un grafo mixto G = (V,E,R) un conjunto V de vértices, un conjunto E de aristas sin paralelos y un multiconjunto R de arcos dirigidos (pueden ser paralelos).
Una instancia de MCDARP es un grafo mixto finito G=(V,E,R), cuyas aristas tiene pesos dados por la función d: $E \rightarrow \mathbf{R}_{\geq 0}$, un numero de transportes $m \in \mathbf{N}$, una capacidad $k \in \mathbf{N}$ y una posición de inicio de los transportes $o \in V$. La red de transporte está representada por E y los objetos a transportar están representados por los arcos R. Cada objeto $r \in R$ debe ser llevado por el transporte del vértice origen $\alpha(r) \in V$ al vértice destino $\omega(r) \in V$.

El movimiento de un transporte del vértice v al w llevando un conjunto de objetos Q se representa como (v,w,Q). (La cantidad |Q| de objetos en un movimiento debe ser menor o igual que la capacidad k)
Una transportación es una serie de movimientos $T = (v_1,v_2,Q_1)\ (v_2,v_3,Q_2)\ ...\ (v_h,v_{h+1},Q_h)$. El costo de la transportación T es $C(T) = \sum_{i=1}^{h} d(v_i, v_{i+1})$ donde $d(v_i, v_{i+1})$ es la menor distancia entre v_i y v_{i+1} en G. Un objeto r es movido por T de v_1 a v_{h+1} si $r \in Q_i$ para $i = 1..h$
Una solución factible cerrada no preemptiva para una instancia de MCDARP es un conjunto de m transportaciones $\{T_1..T_m\}$ tales que:

- T_i empieza y termina en el vértice o

- Para cada T_i existe un $R_i \subset R$ tal que para todo objeto $r \subset R_i$, la subsecuencia de movimientos (v_j,v_{j+1},Q_j) de T_i que cumplen $r \subset Q_j$ es una subsecuencia contigua de T_i y forma una transportación entre el vértice origen y destino de r.
- Para todo j, h ($j \neq h$) entre 1 y m, se cumple que $R_j \cap R_h = \varnothing$ y $\bigcup_{j=1}^{m} R_j = R$.

Definimos el costo de una solución como la suma de los cuadrados de los costos de cada una de sus transportaciones: $K(\{T_1..T_m\}) = C^2(T_1) + ... + C^2(T_m)$

El objetivo del problema MCDARP es encontrar una solución factible tal que tenga el mínimo costo posible.

Teorema: Para toda cantidad de transportes dada, MCDARP cerrado no preemptivo es NP-Completo en grafos generales y de árbol.
Demostración: Usaremos una inducción completa sobre el numero de transportes, asumiendo que es NP-completo para un m dado, y demostrando que también debe ser NP-completo para m+1. Como CDARP es equivalente a MCDARP con m=1, y es NP-completo en ese tipo de grafos, quedará demostrado.

Sea una instancia de MCDARP $I_m = \{(V,E,R),d,m,k,o\}$. Tendrá una solución óptima OPT_m y cuyo costo será $K(OPT_m)$.
Consideremos una instancia idéntica pero con un solo transporte I_1 . Tendrá una solución óptima OPT_1 , de costo $K(OPT_1)$. OPT_1 es una también solución óptima al CDARP correspondiente a I_1 de costo $\sqrt{K(OPT_1)}$.
Podemos hallar una valor estrictamente superior a $K(OPT_1)$ buscando una solución cualquiera de I_1 y alargándola con un ciclo inútil. Este valor entonces cumple que $K(NOPT_1) > K(OPT_1)$

Definamos ahora un $I_{m+1} = \{(V',E',R'),d',m+1,k,o\}$ donde agregamos un transporte t', un vértice v', una arista e'=(o,v') de costo $\sqrt{K(NOPT_1)}/2$ y un objeto r'=(o,v') (o sea, de origen o y destino v'). A notar, esto fuerza que el grafo resultante sea no lineal.
Es obvio que toda solución (óptima o no) de I_{m+1} deberá incluir una secuencia de movimientos de forma S' = (o,v',Q')(v',o,Q") tal que la diferencia entre Q' y Q" es el objeto r' . Esta secuencia tendrá costo $C(S') = \sqrt{K(NOPT_1)}$

Sea $OPT_{m+1} = \{T_1..T_m , T'\}$ una solución óptima de I_{m+1} . Como los transportes son intercambiables, podemos asumir que la secuencia tipo S' está en la transportación T' del transporte t'.

Entonces, $T' = (o,v_2,Q_1)..(v_{j-1},o,Q_j)(o,v',Q')(v',o,Q")(o,v_{j+1},Q_{j+1})..(v_{i-1},o,\varnothing)$
Podemos asumir que r' fue recogido justo al principio de la secuencia tipo S' (si hubiera sido recogido antes, por ejemplo al principio de T', no habría diferencia de costo con una solución en que ocurriera lo primero).
Entonces, $T' = (o,v_2,Q_1)..(v_{j-1},o,Q_j)(o,v', Q_{j+} \cup r')(v',o, Q_j)(o,v_{j+1},Q_{j+1})..(v_{i-1},o,\varnothing)$

Sea $T'' = (o,v_2,Q_1)..(v_{j-1},o,Q_j)(o,v_{j+1},Q_{j+1})..(v_{i-1},o,\varnothing)(o,v', r')(v',o,\varnothing) = (S)(S')$

T'' tendrá el mismo costo que T' pues recorre las mismas aristas. Podemos entonces reempl azar T' por T'' y la solución $\{T_1..T_m , T''\}$ seguirá siendo óptima.

Sea $OPT'_{m+1} = \{T_1..T_m S, S'\}$ otra solución de I_{m+1} .
Se cumple que $K(OPT'_{m+1}) \geq K(OPT_{m+1})$

Entonces tenemos que :
$K(OPT_{m+1}) = C^2(T_1)+.. +C^2(T_{m-1})+C^2(T_m)+ C^2(S)+2C(S)C(S')+ C^2(S')$
$K(OPT_{m+1}) = C^2(T_1)+.. +C^2(T_{m-1})+C^2(T_m)+ C^2(S)+2C(S)\sqrt{K(NOPT_1)} + K(NOPT_1)$

$K(OPT'_{m+1}) = C^2(T_1)+.. +C^2(T_{m-1})+ (C(T_m)+C(S))^2 + C^2(S')$
$K(OPT'_{m+1}) = C^2(T_1)+.. +C^2(T_{m-1})+ C^2(T_m) + C^2(S)+ 2C(S)C(T_m) + K(NOPT_1)$

$K(OPT'_{m+1}) - K(OPT_{m+1}) = 2C(S)(C(T_m) - \sqrt{K(NOPT_1)}) \geq 0$

Pero si $C(T_m) \geq \sqrt{K(NOPT_1)}$ entonces $C(T_m) \geq \sqrt{K(NOPT_1)} > \sqrt{K(OPT_1)}$ lo que implica que el costo de T_m , transportación de un subconjunto de R, es mayor al costo de una transportación entre todos los objetos de R. Eso implica que se puede construir T'_m , transportación de ese mismo subconjunto, de costo menor al de T_m , lo que contradice el que OPT_{m+1} sea una solución óptima de I_{m+1} .

Entonces, necesariamente, C(S) = 0 , por lo que $S = \varnothing$, T''=(S') , y $\{T_1..T_m , S'\}$ es una solución óptima de I_{m+1} .

Pero como $\{T_1..T_m\}$ es solución de I_m , debe ser una de sus soluciones óptimas, pues si no $K(OPT_m, S')$ sería aún menor que $K(OPT_{m+1})$ y OPT_{m+1} no sería óptima.

Es decir que construimos una solución óptima de I_m a partir de una solución óptima de I_{m+1} con manipulaciones de tiempo polinomial.

Lo que implica que si MCDARP con m transportes es NP-completo , entonces también lo será con m+1 transportes, que es lo que queríamos demostrar.

3.4) Estudios Estadísticos sobre el Uso y Capacidad de Ascensores

Otro tipo de estudio sobre los ascensores ha sido no teórico sino estadístico, analizando el comportamiento de auténticos sistemas de ascensores desde su invención hasta el presente día. Esto ha brindado herramientas de base útiles para el diseño de nuevos sistemas y por ende de simuladores.

La primer variable que se estudió fue la llegada de pasajeros. Este estudio brindó a lo largo del tiempo algunas conclusiones muy importantes. Al analizar la distribución de las llegadas respecto al tiempo, se observo que por lo general siguen una distribución de Poisson, pero su frecuencia cambia durante distintos intervalos del día. Más aun, se observó que edificios de categorías similares presentan distribuciones similares. [1] Por ejemplo, en un edificio residencial se observa un flujo de entradas y salidas bastante uniforme durante el día, con algunos incrementos a las horas de la mañana y la noche saliendo y entrando respectivamente. Por el contrario, en el edificio de una universidad se observan notorios picos de demanda a las horas de comienzo y finalización de cada clase.

Este tipo de estudios evidenció un concepto básico en el diseño de los sistemas de ascensores: la necesidad de satisfacer las demandas pico. Según el tipo de edificio, se han observado picos de subida, bajada o subida y bajada simultáneas, pero es ampliamente aceptado que no importa el edificio, el caso más crítico es el del pico de subida. En este caso, se asume que una gran cantidad de la población total del edificio desea ingresar al mismo tiempo, estando originalmente todos los ascensores en la planta baja. Cabe notar que en este caso no hay muchas alternativas para el ruteo: cada ascensor deberá llenarse, subir haciendo las paradas necesarias y volver vacío a la planta baja a recoger más pasajeros. La única decisión de control es cuándo abrir y cerrar las puertas. (Pepyne y Cassandras [12] demostraron que el algoritmo óptimo para minimizar el tiempo media de espera del pasajero hace salir al ascensor de la planta baja cuando su contenido supera un numero t de pasajeros, siendo t variable según la cantidad de ascensores en la planta baja y el ritmo de lleg ada de nuevos pasajeros)

Esta situación fue considerada tan importante que se definieron estándares internacionales para la capacidad mínima de los ascensores en un edificio basados en ella. La idea es que si hay suficientes recursos para satisfacer esta demanda pico de forma razonable, cualquier algoritmo de ruteo que se use tendrá los recursos para responder a todas las demás condiciones posibles.

Dichos estándares están basados en dos variables [2]:

- La capacidad de manejo *hc*, que representa cuantos pasajeros puede llevar el sistema en cinco minutos durante el pico de subida. (La capacidad de manejo relativa muestra el porcentaje de la población del edificio capaz de ser transportada en esos cinco minutos)
- El intervalo *i* , que representa aproximadamente la espera de un usuario en el sistema.

A partir de las formulas para calcular estas variables y de los objetivos mínimos para cada variable es posible determinar la capacidad y cantidad mínima de ascensores necesarios para el edificio.

A continuación definiremos dichas formulas [2]:

Sean: H el piso más alto al que llega el ascensor.
M la cantidad de pasajeros en el ascensor al dejar la planta baja.
C la capacidad máxima del ascensor.
N la cantidad de ascensores en el grupo.
t_v el tiempo para recorrer un piso a velocidad normal.
t_s el tiempo para detenerse en un piso.
t_M el tiempo que baje un pasajero.
S el numero de paradas probables en un viaje de subida ($S = H - H((H-1)/H)^M$ [1])
T el tiempo total de un viaje redondo ($T = 2Ht_v + (S+1)t_s + Mt_M$ en su forma general)

Entonces: $i = \frac{T}{N}$ y $hc = \frac{0.8CN}{T}$

El modificador 0.8 surge del hecho que no es eficiente esperar a que un ascensor se llene antes de hacerlo subir. La práctica arroja que es razonable esperar un promedio de ocupación del 80% de la capacidad máxima.
A notar que para aplicar estas fórmulas se debe normalizar la unidad de tiempo usada. En particular en la formulación final de *hc* la unidad de tiempo debe ser 5 minutos (según la definición estándar de *hc*).

Hay recomendaciones internacionales definidas para edificios residenciales, de oficinas, hoteles y hospitales. Por ejemplo, para residenciales *hc* debe ser mayor al 5% e i estar entre 40 y 100 segundos.

Para el caso de edificios bajos estos valores han sido compilados en tablas de referencia rápida [1].

Finalmente, cabe mencionar que aparte de estos dos parámetros estándar a veces se agregan un tercero y cuarto, menos bien definidos: uno que se refiere al tiempo total del pasajero en el sistema (por ejemplo, tiempo máximo de transito del ultimo pasajero, o tiempo de viaje del ascensor del primer al último piso sin parar) y otro que representa el tiempo promedio de espera a que llegue el ascensor. (en [2] hay tablas definiendo diferentes variables y sus valores recomendados)

4) ALGORITMOS DE CONTROL

4.1) Consideraciones Preliminares

En este capítulo pasaremos revista a diferentes algoritmos y estrategias de control de sistemas de ascensores, preferentemente aquellos que hayan sido utilizados en instalaciones reales. No obstante, la mayoría de las compañías de ascensores mantienen los detalles de sus sistemas de control en secreto, lo que dificulta seriamente una revisión comprehensiva del estado del arte. Además, es conocido que generalmente utilizan mezclas de diversas estrategias según la situación del tráfico, dificultando una clasificación sistemática. Por estas razones, muchos de los algoritmos que presentamos están basados en formulaciones teóricas puras verificadas con simuladores, algunas de las cuales han sido incorporadas en sistemas de control reales, y otras que constituyen investigaciones de punta.

Es claro que los sistemas de múltiples ascensores son el principal objetivo a optimizar, dado que son instalados cuando el volumen de pasajeros es muy grande, y deben satisfacer las demandas más críticas. Por esto la enorme mayoría de los algoritmos que revisaremos están pensados para el control de grupos de ascensores.

Existen estrategias de diseño de sistemas de ascensores que escapan al alcance de esta revisión. Por ejemplo, un recurso común para su simplificación en edificios altos es el zonamiento de los ascensores. Esto consiste en asignar grupos de pisos fijos a diferentes grupos de ascensores. En el malogrado World Trade Center había tres “plantas bajas” (una auténtica y 2 “sky lobbies”) desde donde salían ascensores que servían únicamente a los pisos de esa zona, y ascensores express para comunicar los “lobbies” entre sí [2]. Aquí el zonamiento era fijo desde la construcción del edificio, pero también puede ser estático aunque variable según la hora del día (hay estudios sobre zonamiento dinámico según el tráfico instantáneo [13] pero no conocemos implementaciones). Nos concentraremos en este capítulo en aquellos algoritmos y estrategias que rutean ascensores dentro de un conjunto de pisos a los que sirven equitativamente.

Finalmente, cabe apuntar una distinción importante entre algoritmos de ruteo. Aquellos que cuando asignan una llamada (un pedido externo) a un ascensor no la cambian nunca son llamados algoritmos avaros o “greedy”, a diferencia de los algoritmos que pueden modificar la asignación de una llamada aún no atendida si la situación cambia. Esta distinción es menos trivial de lo que parece: en Japón, por ejemplo, es tradición informar inmediatamente al usuario de cuál ascensor va a responder a su llamada, lo que obliga a usar algoritmos avaros.

4.2) Algoritmos Clásicos

Estos son los primeros algoritmos que se diseñaron para controlar un ascensor o un grupo de ascensores, antes de la invención del control de grupos. [1][2]

- Control no colectivo: Es el más simple de todos. Cuando el ascensor está en movimiento, no responde a ningún pedido, ni interno ni externo. Si recibe un pedido cuando esta inmóvil, lo

atiende. En caso que haya más de un ascensor, responde el que se encuentre más cerca. Es usado hoy en día principalmente por montacargas.

- Selectivo de colas interconectadas (IQS): Se mantiene una cola de llamadas. El ascensor atiende y completa cada llamada antes de responder a la siguiente en orden de antigüedad.
- Colectivo interconectado en descenso (IDC): Cuando el ascensor sube, ignora los pedidos externos, solo atiende los pedidos de parada internos en orden ascendente. Una vez que fue atendido el pedido interno o externo más alto, el ascensor puede invertir la dirección. En la bajada, atiende tanto los pedidos internos como externos en secuencia descendente. Con sensores de carga se puede evitar que el ascensor se detenga ante un pedido externo si ya va lleno. Este algoritmo es apropiado en edificios residenciales donde el tráfico es principalmente entre la planta baja y los pisos superiores.
- Colectivo interconectado completo (IFC): Se debe disponer de dos pulsadores en cada piso, uno para pedir subida y otro para pedir bajada. Cuando el ascensor se mueve en una dirección, atiende todos los pedidos internos que se encuentren en la dirección en la que viaja y todos los externos que pidan ir en esa dirección. Cuando ha terminado de cumplir con ellos, atiende el pedido más extremo que quede en esa dirección y empieza a moverse en la opuesta.
  Cuando este método se aplica para manejar más de un ascensor, aparece el fenómeno de aglutinamiento. Los diferentes ascensores tienden a moverse a los mismos pisos simultáneamente, pues todos atienden las mismas llamadas. Una estrategia corriente para lidiar con este problema fue retener a los ascensores que lleguen a la planta baja y liberarlos a intervalos variables.

Cabe notar que el IFC fue el primer algoritmo al que se le aplicó el control de grupos, manteniéndose la estrategia básica de completar el movimiento en una dirección antes de cambiar pero delegando al controlador la tarea de decidir cuál de entre los ascensores calificados debe atender una llamada externa dada.

4.3) Algoritmos Heurísticos Modernos

Estos algoritmos son similares a los anteriores en que usan reglas simples para obtener una solución satisfactoria a un problema complejo. Aunque muchos sistemas de control auténticos han usado y utilizan algoritmos de este tipo, los aquí mencionados provienen principalmente de estudios académicos. [14]

- Cola Más Larga Primero (LQF): [15] Este algoritmo se centra en la idea de asignar prioritariamente los ascensores a los pedidos externos de bajada que corresponden a las colas de espera más largas. Cuando un ascensor queda libre en un piso bajo, se le asignan todos los pedidos externos de subida no asignados. Cuando queda libre en un piso alto, se lo manda al piso no asignado con la cola más larga de pedidos de bajada, y a partir de ahí atiende los pedidos de bajada que encuentre en su camino.
- Piso Sin Contestar Más Alto Primero Básico (HUFF Básico): [15] Este algoritmo se centra en la idea de asignar prioritariamente los ascensores a los pedidos externos de bajada que corresponden a los pisos más altos. El HUFF Básico se comporta igual que el LQF, excepto que se manda el ascensor al piso más alto con un pedido de bajada no asignado en vez de al que tiene la cola más larga.

- Balance de Carga Dinámico (DLB): [15] Este algoritmo asigna sectores continuos disjuntos del edificio a cada ascensor para mantenerlos separados. Los sectores son calculados de tal manera de balancear sus cargas, y se los recalcula cada vez que cambian las condiciones (no es un algoritmo avaro)
- Algoritmo OTIS Round-Robin (ORR): [15] Es otro algoritmo basado en sectores, similar al de muchos controladores reales. Define sectores estáticos, uno menos que la cantidad de ascensores, y le asigna uno a cada ascensor que deja la planta baja en orden round-robin. Cada ascensor responde únicamente a los pedidos externos de subida y bajada de su propia zona y aparca en ella si no hay llamadas.

Cabe notar que LQF y HUFF Básico fueron diseñados específicamente para optimizar la situación de pico de bajada.

4.4) Algoritmos de Búsqueda de Óptimos Locales

Estos algoritmos se caracterizan por el hecho que todos efectúan una búsqueda más o menos exhaustiva sobre el espacio de soluciones del problema conocido para decidir las asignaciones de llamadas. Se eligen aquellas que corresponden a la solución que optimiza la variable de control usada por el algoritmo.
Decimos el espacio conocido primero porque hay variables que no pueden ser conocidas por un auténtico sistema de ascensores, y segundo porque con cada nuevo pedido el espacio de soluciones varía. La idea (y lo observado en la práctica) es que aún un óptimo local dará un buen resultado al final.
El problema principal radica en que la cantidad de soluciones posibles aumenta exponencialmente con la cantidad de llamadas. Esto causa que la búsqueda general de un óptimo no sea factible en tiempo real para edificios grandes (Ley de Moore mediante, eventualmente esta capacidad debería ser alcanzada, pero aún no estamos allí). Por lo tanto, estos algoritmos deben usar algún método para reducir el espacio de soluciones que buscarán.

- Algoritmo Clásico de Base: [2] Este algoritmo reduce el espacio de soluciones utilizando algunos criterios comunes comprobados en la práctica:
    - Atender todos los pedidos internos secuencialmente.
    - No revertir la dirección hasta atender todos los pedidos internos.
    - No atender pedidos externos que se dirigen en una dirección contraria a la del ascensor
    - Detenerse en un piso sólo si un pasajero quiere subir o bajar (se permiten excepciones, como el aparcar ascensores en períodos de baja demanda)

  Podemos ver que este algoritmo fundamentalmente efectúa una búsqueda en el espacio de soluciones plausibles para el IFC con múltiples ascensores.
  Notemos que en la práctica, casi todos los algoritmos utilizan estos criterios aparte de otros propios.
- Asignación de Llamadas Adaptativa (ACA): [2] Este es un algoritmo avaro, en el que sólo se deciden las asignaciones de nuevas llamadas, partiendo de la base que las que ya fueron asignadas no se modifican. Esto reduce considerablemente el espacio de soluciones a buscar, pero se corre el riesgo que las asignaciones más viejas pierdan su optimalidad.

- Algoritmo con Ruteo Óptimo (OR): [2] Este algoritmo revierte el concepto del IFC para hacer las asignaciones. En vez de buscar los pedidos externos más cercanos en la dirección de movimiento para asignarlos al mejor ascensor, empieza por asignar los pedidos más lejanos y procede hacia los más cercanos. Luego vuelve a efectuar la asignación usando la información obtenida en la iteración anterior, hasta que el resultado no varíe. Las simulaciones han arrojado buenos resultados, a pesar que ese algoritmo no siempre encuentra el verdadero óptimo.
- Piso Sin Contestar Más Alto Primero Modificado (HUFF): [15] Este algoritmo es un refinamiento sobre el HUFF Básico. Cuando hay ascensores libres en un piso alto, se les asignan pisos a partir de la minimización de una función de carga. La misma divide el edificio en sectores variables y le asigna a cada ascensor libre el piso más alto de cada sector, eligiéndose la división que minimiza el cuadrado de los tiempos de espera. Cuando queda libre en un piso bajo, se comporta igual que en el HUFF Básico. No es avaro, pues la asignación de cada ascensor pueden cambiar siempre y cuando éste no haya llegado aún a su piso asignado.
- Minimización de Intervisitas Finitas (FIM): [15] Es similar al HUFF, pero puede asignar tanto pedidos de subida como de bajada a todos los ascensores, no sólo a los libres. Además, puede asignar varios pedidos distintos a cada ascensor (o sea, divide el edificio en sectores y le puede asignar varios sectores no contiguos a un mismo ascensor).
- Algoritmo de Vaciado del Sistema (ESA): [15][14] Este algoritmo es similar al del FIM, pero el objetivo de su función de carga es minimizar la suma de los tiempos que faltan para que cada pasajero sea recogido. A todos los efectos, su objetivo es encontrar la manera más rápida de vaciar el sistema (o sea, atender a todos los pedidos) asumiendo que no llegarán más pasajeros. ESA utiliza información sobre el largo de las colas de espera que no estaría disponible en un sistema de ascensores auténtico. Otra versión, el ESA/nq utiliza información estadística sobre la frecuencia de llegadas para estimar los largos de las colas.
- Algoritmo de Despacho de Estimación Concurrente (CEDA): [16] A partir de su demostración matemática de un algoritmo de control óptimo para el pico de subida [12], Pepyne y Cassandras implementaron un controlador que utiliza algoritmos online para resolver este problema. El momento en que se habilita cada ascensor para llevar pasajeros de la planta baja a los pisos superiores depende de que la cantidad de pasajeros que hay en él supere un cierto nivel de activación (este valor se puede estimar a partir de los sensores de peso o entrada/salida de los ascensores). El nivel de activación varía según el tiempo de espera de los pasajeros en el sistema, valor que puede ser estimado usando varias estrategias. La optimización se hace sobre períodos discretos de un día, atendiendo cada uno con un nivel arbitrario, buscando el nivel óptimo con el que ese período habría sido atendido, y usando dicho nivel para atender ese mismo período al día siguiente.

Cabe notar que FIM, ESA y HUFF fueron diseñados específicamente para optimizar la situación de pico de bajada, mientras que CEDA lo fue para la situación de pico de subida

4.5) Algoritmos de Búsqueda de Óptimos Locales con Estimación Futura

Los anteriores algoritmos sufrían del problema fundamental que no hay garantías que el óptimo local hallado en un ciclo de asignaciones siga siendo óptimo cuando llegue el próximo pedido.

Estos algoritmos intentan paliar ese problema simulando posibles pedidos en un futuro cercano y eligiendo la solución que resulte más ventajosa para el conjunto de posibilidades más significativo.

- OPTICON: [2] Es similar a los sistemas de decisión de los programas para jugar ajedrez. En cada ciclo se decide si un ascensor debe detenerse, arrancar, continuar su movimiento, abrir sus puertas o cerrarlas. El algoritmo calcula todas las posibles secuencias de decisiones en un futuro acotado. Para probarlas, se simulan varias instancias de tráfico futuro generando llegadas y pedidos al azar siguiendo distribuciones conocidas. Las asignaciones en cada instancia se hacen según algún criterio conocido como el IFC. Se calcula el costo promedio de cada instancia y se elige la secuencia de decisiones que arrojó menor costo.
  Cabe notar que este algoritmo no usa los nuevos pedidos como momento de decisión. Esto puede causar movimientos erráticos de los ascensores, sobre todo cuando hay poco tráfico.
- Asignación de Llamadas Adaptativa Dinámica (DACA): [2] Como el ACA, es un algoritmo avaro que no modifica asignaciones una vez hechas. Cuando llega un nuevo pedido, se prueba cual sería el resultado si se lo asignara a cada ascensor. Para esto se genera una corta simulación de tráfico futuro, y se asigna ese tráfico con ACA (aunque cualquier otro algoritmo serviría). Luego de esto se asigna el pedido al ascensor cuya simulación acarreó menor costo.

4.6) Sistemas Expertos y Lógica Difusa

Los sistemas expertos fueron la primera tecnología de Inteligencia Artificial aplicada a los sistemas de control de grupos de ascensores. La idea era que grupos de expertos trabajaran juntos para definir cuales eran las mejores respuestas posibles del sistema ante los diferentes patrones de tráfico posible. Una vez codificado ese conocimiento en el sistema experto (que originalmente no necesitaba ser más complejo que una planilla electrónica) el sistema detectaría el patrón del tráfico y ajustaría su comportamiento. Había algunos problemas con este sistema, notablemente la necesidad de definir objetivos claros para los expertos. [3].

Los sistemas expertos puros fueron usados para optimizar diversos aspectos del control de ascensores, pero eventualmente todos sufrían por la incertidumbre y la incompletitud de los datos disponibles. Por esta razón, se los combinó con sistemas de lógica difusa, diseñados para manejar variables inciertas. Estos sistemas servirían de interfase entre los sensores de tráfico y la base de conocimiento experto, mejorando el rendimiento.

- Mitsubishi AI-21xx: [4] La compañía Mitsubishi fue la primera en aplicar lógica difusa al problema del ruteo de ascensores a finales de los 80s. Hoy en día los sistemas expertos con lógica difusa siguen siendo centrales para los sistemas de asignación de llamadas de sus controladores más avanzados.
- KONE TSM9000: [2] Este controlador de la compañía KONE utiliza lógica difusa para identificar 26 tipos distintos de tráfico y activar las acciones de control correspondientes.
- Sistema experto de lógica “defeasible”: [18] En este trabajo se describe como usar un modelo “defeasible” (donde las reglas tienen un orden de precedencia) para manejar el ruteo de un ascensor, y su implementación en el lenguaje d-Prolog.

- Controladores difusos: [19] En este trabajo se describe un controlador de grupos de ascensores basados puramente en lógica difusa, con énfasis en la estructura del diseño y el uso de diferentes conjuntos de reglas para diferentes tipos de tráfico.

4.7) Redes Neurales

Las redes neurales artificiales son un paradigma de procesamiento de información inspirado en la estructura de los cerebros de los mamíferos. Su elemento central es un conjunto de elementos de procesamiento individuales pero altamente interconectados a través de conexiones ponderadas (emulando neuronas y sinapsis). Un algoritmo de ajuste de los pesos de las conexiones permite a la red aprender a resolver mejor un problema dado. Hay muchos tipos de redes neurales clasificados según su estructura y tipos de algoritmos de aprendizaje.
Estos sistemas han sido usados con éxito para atacar problemas demasiado complejos para ser resueltos vía soluciones algorítmicas clásicas. Es por lo tanto claro que el control de grupos de ascensores es un candidato natural a ser manejado por ellos, y desde el principio de los 90s han aparecido estudios e implementaciones a estos efectos. Han variado desde implementar los sistemas de lógica difusa como redes neurales hasta definir todo el controlador como una red neural, permitiendo que se personalice para cada edificio particular a través del aprendizaje.

- Mitsubishi AI-21xx: [4] Estos controladores utilizan una red neural para identificar los patrones de tráfico que luego son informados a un sistema de lógica difusa.
- Redes de Aprendizaje Reforzado: [5][14] Estos trabajos analizan el uso de redes neurales con algoritmos de aprendizaje reforzado para controlar los ascensores, ya sea con redes compartidas o disjuntas (una para cada ascensor). En particular comparan los resultados obtenidos en simuladores con los de algoritmos heurísticos conocidos, y verifican que se obtienen mejores resultados.
- Redes Forward-Feeding: [20] Este trabajo diseña una única red neural de tres niveles para encargarse del ruteo colectivo de los ascensores. Compara los resultados obtenidos con simuladores con los de un controlador difuso y verifica que se obtienen mejores resultados.

4.8) Algoritmos Genéticos

Este tipo de algoritmos está basado en el concepto de evolución vía selección natural. En su forma más simple, se genera mas o menos al azar una serie de programas para resolver un problema y se los prueba. Los programas que resultaron menos exitosos son descartados y se vuelve a poblar el conjunto de programas con mutaciones y cruzas de los sobrevivientes. Se continúa así hasta encontrar un programa eficiente (o sea, capaz de sobrevivir varias generaciones). Uno de los problemas a los que se lo ha aplicado es a hallar el ruteo óptimo de los ascensores para las llamadas existentes [2], aunque no conocemos ninguna aplicación comercial.
Un trabajo en dicha área es el de Gudwin y Gomide [17], en el que describen cómo se puede codificar un sistema de control de eventos discretos en forma de algoritmo genético, y aplican esa teoría a un modelo simplificado de grupos de ascensores. Fundamentalmente, cada asignación de un pedido externo a un ascensor representa un gene, cada cromosoma consiste de un gene por cada pedido, y un conjunto de cromosomas generados al azar compiten y se cruzan entre sí para

minimizar el tiempo promedio de espera. Luego de varias generaciones, se elige el cromosoma más apropiado y se asigna el pedido correspondiente a su primer gen.

4.9) Sistemas de Planeamiento

Otro campo de la inteligencia artificial son los sistemas de planeamiento. El concepto gira alrededor de crear sistemas de planeamiento versátiles, capaces de atacar cualquier problema definido dentro de un dominio especificado cuya solución se pueda especificar como una secuencia de acciones (o plan). Hay muchas estrategias y tecnologías para resolver problemas de planeamiento, pero se tiende a enfatizar el uso de lenguajes estándar de especificación de problemas y dominios para aumentar la compatibilidad entre las diferentes implementaciones.

La ventaja de estos sistemas es que están comercialmente disponibles y todo lo que haría falta sería definir el dominio del ascensor y el problema para obtener una solución, y un cambio de especificación no requeriría cambiar el software. Algunos trabajos como el de Koehler y Shuster [21] han estudiado el problema de redactar dichas definiciones, pero los resultados no han sido demasiado satisfactorios a causa de limitaciones del lenguaje de especificación PDDL. Sin embargo, en los últimos años han habido mejoras en los lenguajes de especificación, y es de esperar que una definición útil del problema de control de ascensores como un problema de planeamiento sea realizable a corto plazo.

Un evento de interés en este campo es la International Planning Competition (IPC), una competencia bianual asociada a las Artificial Intelligence Planning and Scheduling Conferences, donde diversos sistemas de planeamiento compiten para resolver una serie de problemas estándar.

5)DISEÑO DE UN SIMULADOR

5.1) Antecedentes y Justificación

Como hemos visto en capítulos anteriores, los simuladores son una herramienta que ha sido muy utilizada en el diseño de sistemas de ascensores, tanto para el planeamiento de edificios específicos como para el testeo de algoritmos de control. A medida que la electrónica ha avanzado, permitiendo sistemas de ascensores más complejos, también simuladores mas rápidos y perfeccionados se han vuelto posibles. Con los avances realizados en la computación personal en la última década, la capacidad de realizar un simulador íntegramente como software ejecutándose en un microprocesador está al alcance del público general.

Es por esta razón que llama la atención la escasa cantidad de software de simulación de ascensores disponible. Las compañías de ascensores mantienen sus algoritmos de ruteo y control confidenciales, por lo que es razonable que los simuladores en que los ejecutan también lo sean. Menos comprensible es que se dé un fenómeno similar en el ámbito académico. Prácticamente cada trabajo que sugiere o analiza un algoritmo utiliza un simulador para probarlo[5][14][17][19][12][16][15] , pero por lo regular se trata de un simulador escrito específicamente para esa ocasión, y cuyo código no es hecho público. La única referencia a un sistema de simulación de ascensores genérico en la literatura reciente es el VSE creado por Zong Qun et al [22].

Dado el claro interés de una herramienta de simulación lo más general posible, decidimos dar un primer paso en esta dirección creando una versión 1.0 del programa que llamaremos SSA.
En el mismo incorporaremos algunas funcionalidades que no se encuentran en ningún otro sistema del que tengamos conocimiento, en particular soporte para simular dos fenómenos muy comunes en el mundo real pero subrepresentados la literatura. Los mismos son *Balking* (es cuando un pasajero potencial decide no usar el ascensor si la cola es muy larga) y *Reneging* (es cuando un pasajero decide dejar de esperar el ascensor luego de un cierto tiempo)

Está claro que cuando existen varias aplicaciones capaces de brindar una misma funcionalidad, es deseable que sean capaces de entender sus respectivos formatos de datos. Por lo tanto, otro objetivo que intentaremos alcanzar es definir una especificación de entrada de d atos que sea lo más general y versátil posible, con la esperanza que pueda ser expandida y reutilizada en futuros sistemas de simulación de ascensores.

5.2) Especificaciones y Diseño

5.2.1) Lenguaje Utilizado

El SSA fue escrito en ModSim III, un lenguaje orientado a objetos basado en C++ diseñado específicamente para simulación.
Las razones para la elección de este lenguaje fueron primariamente su disponibilidad inmediata y el hecho que nos era familiar por trabajos anteriores. Estas consideraciones superaron los inconvenientes potenciales del lenguaje. Uno de ellos es el hecho que ha sido discontinuado por sus dueños. El otro es que aunque la característica más distintiva del ModSim III es que permite

simulación orientada a procesos, esto no nos es útil en este caso, pues los algoritmos de control de grupos de ascensores son todos orientados a eventos.

5.2.2) Formato de Entrada

Como dijimos anteriormente, queríamos que nuestro formato de entrada fuera lo más versátil posible, potencialmente capaz de representar la información necesaria para el funcionamiento de cualquier simulador. Para lograrlo, era necesario que fuera capaz de representar los elementos comunes a todo simulador, es decir, la realidad de un edificio con sus ascensores, en todas sus variantes.

Es sin embargo imposible alcanzar semejante nivel de completitud, aunque más no sea porque al pasar el tiempo aparecerán nuevas alternativas no contempladas. Decidimos entonces utilizar la experiencia obtenida durante nuestra investigación del estado del arte para identificar los elementos más significativos de un sistema de ascensores, los que incluiríamos en la especificación. Otros elementos de los que no tuviéramos conocimiento o que aparecieran en el futuro deberían poder ser incluidos expandiendo la especificación. Este requerimiento de expansibilidad, combinada con los de generalidad y compatibilidad nos llevó a la decisión de utilizar el lenguaje XML para escribir nuestra entrada de datos, y a especificarla a través de un esquema XSML.

A continuación describiremos los elementos de nuestro modelo de sistema de ascensores, y como lo representamos en el formato de entrada.

5.2.2.1) Modelo

Inicialmente decidimos limitar el problema haciendo algunas simplificaciones de base. Estas son:

- Nos limitamos a sistemas de ascensores de pasajeros, no incluyendo los de carga.
- Todos los pasajeros son idénticos.
- Las características temporales del sistema son cíclicas, con periodicidad finita.
- Los ascensores son independientes, más allá del sistema de control; no consideraremos específicamente casos como ascensores que comparten el mismo ducto físico u otras estrategias similares.

Identificamos entonces tres elementos principales a modelar: los pisos del edificio, los ascensores, y los pasajeros.

De los pisos nos interesa abstraer su cantidad, su secuencia, y las distancias entre cada piso. Es también importante poder individualizar la planta baja, pues la mayoría de los algoritmos le asigna un papel especial. Esto se puede representar como una secuencia de pisos numerados (la planta baja siempre con numero cero) de los cuales conocemos su distancia al piso siguiente.

De los pasajeros nos interesa saber cuándo van a tomar un ascensor, en qué piso lo hacen, y a qué piso se dirigen. También queremos saber con cuanta gente esperando efectuarán Balking, y con que espera Reneging.
Es prácticamente imposible contar con información exacta del tráfico de pasajeros en un edificio, así que necesariamente debemos recurrir a estadísticas.
Está ampliamente establecido en la literatura que la distribución de Poisson (o su equivalente inversa, la distribución exponencial) es adecuado para representar las llegadas a un sistema de colas de estas características. Por lo tanto podemos considerar que las llegadas de pasajeros a un piso

para tomar un ascensor están dadas por dicha distribución, con tasas de llegada que varían según el tiempo.
Por otro lado, el destino de un pasajero desde un piso está acotado al conjunto de los otros pisos, así que es representable con una distribución discreta sobre ese conjunto, con pesos que varían según el tiempo.
En cuanto a en qué piso el pasajero toma el ascensor hay dos alternativas. Una es individualizar cada pasajero desde que llega al edificio en la planta baja y seguirlo en sus movimientos de piso a piso hasta que sale del edificio. Este método es muy fiel a la realidad, pero extiende la información contenida en el modelo más allá del foco del sistema de ascensores. La otra alternativa es considerar que los pasajeros entran al sistema en cualquier piso, van a su piso de destino y salen al llegar a él. Elegimos usar esta segunda alternativa pues se circunscribe mejor al alcance del modelo y es más versátil. En efecto, usando tasas de llegada apropiadas se pueden aproximar ciclos de entrada y salida de pasajeros como el descrito en la alternativa anterior, pero además se pueden crear situaciones especiales, como picos de subida o de bajada, lo que no sería posible con la primer alternativa.
En lo que respecta a Balking y Reneging, la lógica lleva a pensar que los valores límite deben de seguir una distribución normal o similar, y variar en el tiempo y para cada piso. No obstante, dada la falta de literatura al respecto, simplificaremos ambos valores como constantes permanentes en el tiempo y el espacio.
Por lo tanto, y según las decisiones anteriores, habrá una serie de tasas de Poisson de generación de pasajeros y de distribuciones de destinos asociadas a cada piso del edificio. El ciclo de movimientos en cada piso será independiente del de los otros pisos, y quedará dividido en un conjunto de intervalos, cada uno con su propia tasa de generación y distribución de destinos.

Finalmente, de cada ascensor queremos conocer su capacidad, su velocidad en toda circunstancia, y en qué pisos para. También nos interesa el tiempo que toma subir o bajar del ascensor a cada pasajero.
Definiremos la capacidad como la cantidad máxima de pasajeros que puede llevar el ascensor.
Los pisos de parada son significativos para representar casos como ascensores pares e impares, o pisos prohibidos a ciertas horas. Por esto, para cada ascensor se dividirá el ciclo en intervalos, cada uno con una lista de pisos en los que puede parar.
En cuanto a la velocidad de movimientos del ascensor, decidimos para simplificar identificar tres valores: la velocidad de crucero, el tiempo de arranque y el tiempo de parada. En estos últimos combinaremos los tiempos de cerrar y abrir las puertas respectivamente con la diferencia entre el tiempo real que toma al ascensor acelerar o decelerar completamente y el tiempo que le llevaría recorrer la distancia de arranque o parada a la velocidad de crucero. De esta manera, podemos calcular el tiempo de ir de un piso y parar en otro como el tiempo de arranque más el de parada más la velocidad multiplicada por la distancia entre los pisos. Cabe notar que esta aproximación tiene algunas falacias, a saber que un ascensor muy veloz bien puede no llegar a su velocidad de crucero antes de tener que parar en otro piso.
Finalmente, en lo que respecta a los tiempos de subida y bajada de pasajeros, en la literatura hay referencias a varias distribuciones como la Normal o la de Erlang. Decidimos definirlos entonces como una media y una varianza, con las que se pueden representar varias distribuciones.

5.2.2.1) Representación XML

Definimos dentro del archivo correspondiente a cada edificio el nombre del edificio y el largo en segundos del ciclo de generación de pasajeros, además de un conjunto de elementos tipo piso y un conjunto de elementos tipo ascensor.

Cada elemento piso tiene un numero único (0 si es la planta baja), un nombre (opcional), la distancia en metros al piso inmediato superior (0 si es el último piso) y un conjunto de generadores de pasajeros. Estos generadores dividen al ciclo en intervalos, y constan cada uno del segundo de inicio de ese intervalo dentro del ciclo (el inicio del primero debe ser cero; el final de cada intervalo es el inicio del siguiente o el segundo de fin del ciclo), de una frecuencia de llegada de pasajeros (la frecuencia media de una distribución exponencial; ponemos cero en el primero si no se generan pasajeros en ese piso) y de un conjunto de pesos correspondientes a los demás pisos que representan las probabilidades relativas de que el destino de los pasajeros sean esos pisos.

Por su parte, cada elemento ascensor tiene un número único, un nombre opcional, una capacidad (entera), una velocidad (en metros por segundo, mayor que cero), una media y varianza del tiempo de subida de un pasajero, el tiempo de parada, el tiempo de arranque, y un conjunto de paradas posibles. Dicho conjunto divide al ciclo en intervalos, y cada parada consta del segundo de inicio de ese intervalo dentro del ciclo (el inicio del primero debe ser cero; en final de cada intervalo es el inicio del siguiente o el segundo de fin del ciclo), y de la lista de los pisos en que ese ascensor puede parar en ese intervalo.

La definición formal del formato está dada por el esquema XML EdificioSim.xsd (ver apéndice C).

5.2.3) Estructura del Programa

El programa SSA está divido en módulos, cada uno con una función específica. Los mismos son:

- *Main*: Maneja el loop principal del programa.
- *Windobj*: Maneja el interfaz gráfico del programa. Eso incluye los diálogos de configuración de parámetros dinámicos tales como la cantidad de intervalos de monitoreo, el algoritmo de scheduling a utilizar, las semillas de los generadores aleatorios y los valores límite para Balking y Reneging. Las definiciones de las imágenes que componen el interfaz están en un archivo auxiliar llamado *ascensor.sg2*.
- *XMLmod*: Maneja la entrada y salida de datos en formato XML. Incluye funciones escritas en C++ que sirven de interfase con la biblioteca TinyXml - un parser XML con funcionalidades mínimas, de uso libre, escrito por Lee Thomason (ver la página `http://www.sourceforge.net/projects/tinyxml`)
- *Monitor*: Recolecta y almacena información estadística durante la corrida de la simulación, y formatea los datos de salida. (ver siguiente sección para más información)
- *Scheduler*: Implementa los distintos algoritmos de ruteo del sistema de ascensores.
- *Dataobj*: Implementa las estructuras de datos de los elementos del modelo y sus interacciones mutuas y con el manejador de ascensores.

Estos dos últimos módulos son el núcleo del programa, por lo que los examinaremos con más detalle.

El módulo Dataobj define tres objetos principales que se inicializan con los datos de entrada de un edificio. A continuación mostramos sus elementos y métodos más relevantes.

PISO

```
numero:INTEGER              {numero del piso}
nombre:STRING;              {nombre del piso}
distanciaSig:REAL;          {distancia al siguiente piso}
distanciaPrev:REAL;         {distancia al piso previo}
pasajerosSuben:QueuePas;    {cola de los pasajeros que quieren subir}
pasajerosBajan:QueuePas;    {cola de los pasajeros que quieren bajar}
tasasGen:RankQueueTasa;     {para generar las llegadas al piso}
tasaGenActual:TasaGeneracion;
```

```
TELL METHOD crearNuevoPasajero();
ASK METHOD tiempoProximoPasajero(IN tiempoinicial:REAL; IN inidiact:REAL; IN
tasgenact:TasaGeneracion):REAL;
ASK METHOD startGenPasajeros();
ASK METHOD disposePasajeroBalk(IN direccion:INTEGER; IN pas:Pasajero);
TELL METHOD disposePasajeroReneg(IN direccion:INTEGER; IN pas:Pasajero);
```

PASAJERO

```
pisoDestino:INTEGER;        {piso donde se baja}
tiempoCreacion:REAL;        {instante de origen}
intervaloCreacion:INTEGER;  {intervalo de origen}
actualizaReportes:BOOLEAN;  {se utilizan para Balking y Reneging}
schedDestructor:ACTID;
```

ASCENSOR

```
numero:INTEGER              {numero del ascensor}
nombre:STRING;              {nombre del ascensor}
capacidad:INTEGER;          {capacidad}
velocidad:REAL;             {modificador de velocidad basico}
tiempoSubidaMed:REAL;       {Media del tiempo de subida o bajada de un pasajero}
tiempoSubidaVar:REAL;       {Varianza del tiempo subida o de bajada de un pasajero}
tiempoParada:REAL;          {tiempo para desacelerar y parar en un piso}
tiempoArranque:REAL;        {tiempo para salir de un piso y acelerar}
direccion:INTEGER;          {-1 si baja, 1 si sube, 0 si esta quieto}
pasajeros:BTreePas;         {arbol de los pasajeros en el ascensor}
pisosParada:RankQueuePisosParada;
pisoActual: Piso;           {piso donde se encuentra actualmente}
seMueve:BOOLEAN;            {TRUE si se esta moviendo en este instante, FALSE si no}
```

```
TELL METHOD goToPiso(IN P:Piso);
TELL METHOD stopAtPiso(IN P:Piso);
TELL METHOD subeBajaPas(IN dir:INTEGER);
TELL METHOD subePas(IN dir:INTEGER);
TELL METHOD subeUnPas(IN dir:INTEGER);
TELL METHOD bajaPas(IN dir:INTEGER);
TELL METHOD goAndStopAtPiso(IN P:Piso);
TELL METHOD goStopAndSubeBajaAtPiso(IN P:Piso;IN dir:INTEGER);
```

Los métodos que mencionamos del objeto Ascensor son los que se utilizan para interactuar con el módulo Scheduler.

En dicho módulo se definen los diferentes manejadores como objetos descendientes del objeto básico SchedulerObj. Antes de iniciar la corrida, se elige cuál de estos descendientes se utilizará, y se inicializarán sus valores apropiadamente. En el apéndice B explicamos cómo agregar nuevos manejadores.

SchedulerObj	
nombre:STRING;	{nombre del manejador}
generador:RandomObj;	{generador aleatorio en caso que lo necesite}
maxQueueBalk:INTEGER;	{límite de Balking, si contempla Balking}
maxWaitReneg:REAL;	{límite de Reneging, si contempla Reneging}

ASK METHOD llegoPasajeroPiso(IN P:Piso; IN direccion:INTEGER; IN pas:Pasajero);
ASK METHOD finGoToPiso(IN A:Ascensor; IN P:Piso);
ASK METHOD finStopAtPiso(IN A:Ascensor; IN P:Piso);
ASK METHOD finSubeBajaPas(IN A:Ascensor; IN P:Piso; IN dir:INTEGER);
ASK METHOD finSubePas(IN A:Ascensor; IN P:Piso; IN dir:INTEGER);
ASK METHOD finSubeUnPas(IN A:Ascensor; IN P:Piso; IN dir:INTEGER);
ASK METHOD finBajaPas(IN A:Ascensor; IN P:Piso; IN dir:INTEGER);
ASK METHOD finGoAndStopAtPiso(IN A:Ascensor; IN P:Piso);
ASK METHOD finGoStopAndSubeBajaAtPiso(IN A:Ascensor; IN P:Piso; IN dir:INTEGER);

Los métodos básicos definidos por SchedulerObj que mencionamos más arriba deben ser modificados apropiadamente, pues son los que definen el interfaz con los objetos del módulo Dataobj. Aparte de estos elementos y métodos básicos, cada descendiente de SchedulerObj debe definir los suyos propios que sean apropiados para implementar su algoritmo. En general, todo manejador debería tener algún tipo de registro del estado del sistema y métodos de decisión que utilice para rutear los ascensores. (Cabe mencionar que de ser necesario, los objetos del módulo Scheduler pueden acceder directamente a la información de los objetos de Dataobj. Esto no es recomendable por una cuestión de prolijidad, pero se dejó la posibilidad abierta para ampliar las posibilidades del desarrollador de nuevos manejadores.)

El funcionamiento del interfaz es el siguiente:

llegoPasajeroPiso es invocado por *crearNuevoPasajero* de Piso para informar al manejador de la llegada de un pasajero a un piso. El manejador puede invocar en ese momento al *disposePasajeroBalk* del piso en cuestión para eliminar al pasajero si el largo de la cola supera el límite de Balking. También puede programar que se ejecute *disposePasajeroReneg* luego del tiempo límite de Reneging (solo se ejecuta si el pasajero aún no subió a un ascensor después de transcurrido ese intervalo). Finalmente, debería activarse una reevaluación del plan de ruteo (pues

el estado cambió), que eventualmente llamará a alguno de los métodos de control de ascensores mencionados en el objeto Ascensor.

Estos métodos de control de ascensores le ordenan a un ascensor ir a un piso, detenerse y/o bajar y subir uno o todos los pasajeros posibles. Sus nombres son explicativos de su función precisa. Luego que el ascensor cumplió con la orden recibida con el métodode control *X*, invocará al método del scheduler *finX*. De esta manera informará al manejador que la orden fue cumplida y que el estado del sistema cambió, lo que debería activar un reevaluación del plan de ruteo de ascensores.

Como los métodos *finX* sólo pueden ser invocados por sus equivalentes *X* del objeto Ascensor, que a su vez sólo pueden ser invocados por el manejador, no hace falta reescribir todos los *finX* al implementar un nuevo manejador. Sólo hace falta modificar los que se vayan a usar, pues sabemos que los otros nunca serán invocados.

5.2.4) Datos Recolectados y Formato de Salida

Como ya mencionamos, al iniciar una corrida podemos subdividir el ciclo de llegada de pasajeros en intervalos regulares. Las estadísticas que recolectaremos se acumularán tanto globalmente para todo el ciclo como localmente dentro de cada intervalo del ciclo.

Los datos que recolectamos en el módulo *Monitor* son:

- Generales:
  - Nombre del Edificio
  - Cantidad de Intervalos
  - Semilla Randómica para la generación de pasajeros
  - Duración total de la corrida
  - Duración del ciclo
  - Algoritmo de Scheduling utilizado
  - Semilla Aleatoria para el scheduler
  - Largo de Cola Crítico para el Balking (si el algoritmo usa Balking)
  - Tiempo de Espera Crítico para el Reneging (si el algorimo usa Reneging)

- Para los pasajeros:
  - Espera Media: Es la media de los tiempos que espera cada pasajero desde que llega hasta que sube al ascensor. En cada intervalo se promedian las esperas de los pasajeros que llegaron en ese intervalo.
  - Espera Cuadrada Media: Es la media de los cuadrados de los tiempos que espera cada pasajero desde que llega hasta que sube al ascensor. En cada intervalo se promedian las esperas cuadradas de los pasajeros que llegaron en ese intervalo.
  - Tiempo Sistema Medio: Es la media de los tiempos que espera cada pasajero desde que llega hasta que se baja del ascensor. En cada intervalo se promedian los tiempos de los pasajeros que llegaron en ese intervalo.
  - Tiempo Sistema Cuadrado Medio: Es la media de los cuadrados los tiempos que espera cada pasajero desde que llega hasta que se baja del ascensor. En cada intervalo se promedian los tiempos cuadrados de los pasajeros que llegaron en ese intervalo.

- Espera Máxima: Es el tiempo de espera máximo que tuvo ningún pasajero durante la corrida. En cada intervalo se recoge la espera máxima de los pasajeros que llegaron en ese intervalo
- Porcentaje Pasajeros Pasados de *n*: Es el porcentaje del total de pasajeros que tuvieron esperas mayores a la cantidad *n* de segundos (definida al iniciar la corrida). En cada intervalo se calcula ese porcentaje sobre los pasajeros que llegaron en ese intervalo.

- Para cada ascensor:
  - Ocupación Media: Es la media ponderada en el tiempo de la cantidad de pasajeros dentro del ascensor. En cada intervalo se calcula la media durante la duración de ese intervalo.
  - Cantidad de Paradas: Es la cantidad de veces que el ascensor se detuvo. En cada intervalo se suma la cantidad de veces que el ascensor se detuvo durante ese intervalo.

- Para cada piso:
  - Largo Cola Medio: Es la media ponderada en el tiemp o de la cantidad de pasajeros que están esperando un ascensor en ese piso. En cada intervalo se calcula la media durante la duración de ese intervalo.
  - Largo Cola Máximo: Es el largo máximo que tuvo la cola de espera en ese piso. En cada intervalo se registra el largo de cola máximo durante ese intervalo.
  - Cantidad Pasajeros Generados: Es la cantidad total de pasajeros que se generaron en ese piso durante la corrida. En cada intervalo se acumulan los pasajeros generados en ese piso durante ese intervalo.
  - Cantidad Pasajeros Balked: Es la cantidad de pasajeros que sufrieron Balking durante la corrida en ese piso. En cada intervalo se acumulan los pasajeros que sufrieron Balking en ese piso durante ese intervalo. Estos valores solo se registran si el algoritmo contempla Balking.
  - Cantidad Pasajeros Reneged: Es la cantidad de pasajeros que sufrieron Reneging durante la corrida en ese piso. En cada intervalo se acumulan los pasajeros generados en ese intervalo que sufrieron Reneging en ese piso. Estos valores solo se registran si el algoritmo contempla Reneging.

Estos datos pueden ser grabados a un archivo de salida en formato XML, estructurado según la anterior descripción. El formato de este archivo está definido por el esquema EdificioRep.xsd (ver Apéndice D)

Algunos de estos valores son graficados en el interfaz del simulador. En la figura 6.2.4.1 vemos los valores de pasajeros que son graficados dinámicamente a medida que se ejecuta la corrida. Tenemos la Espera Media, la Espera Cuadrada Media, el Tiempo Sistema Medio y la Espera Máxima, tanto sus valores globales como la gráfica por intervalos.

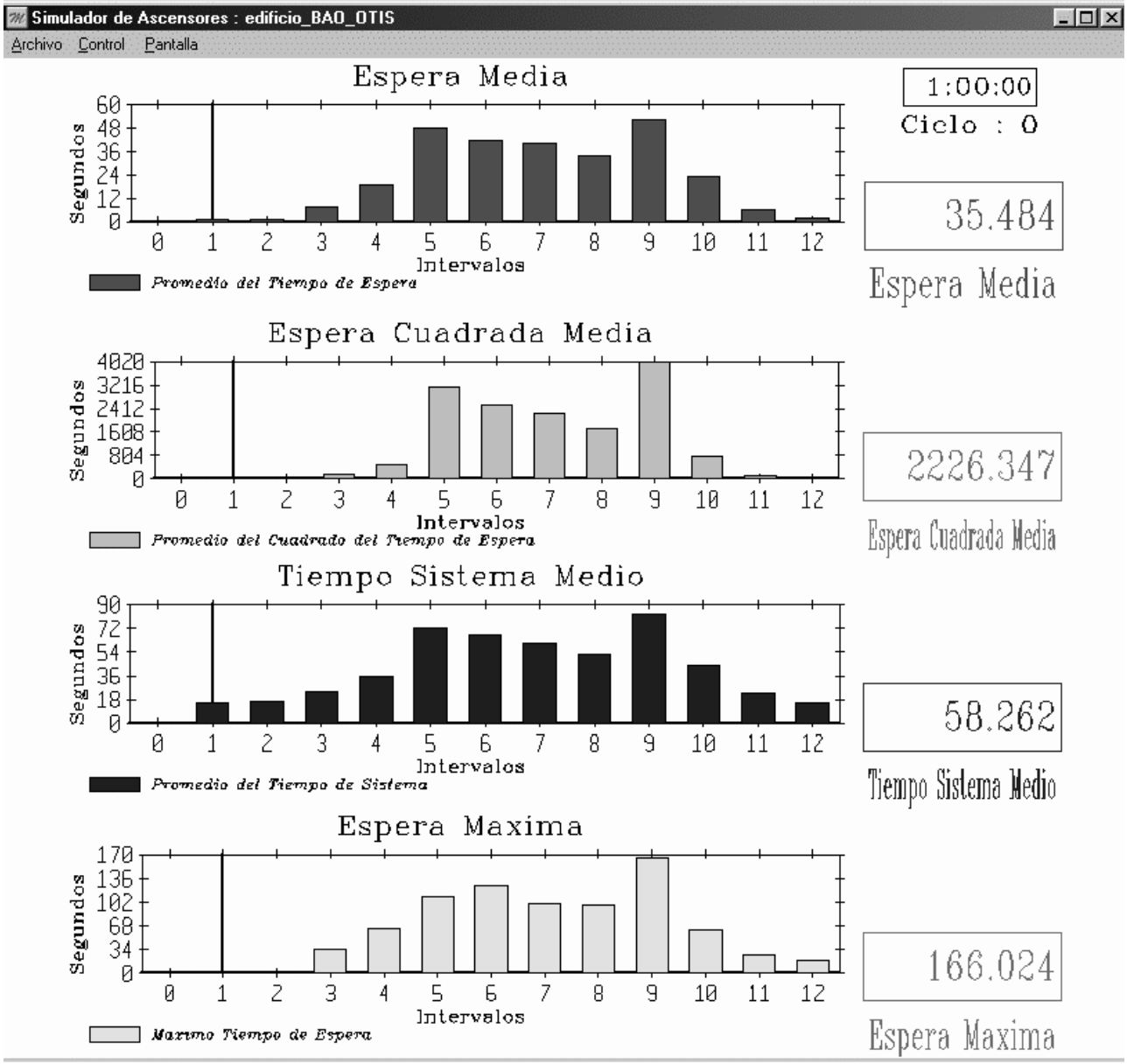


Figura 6.2.4.1

Por otro lado, algunos valores también son graficados pero solo pueden ser vistos al finalizar la corrida. Los podemos ver en la figura 6.2.4.2. Estos son el Porcentaje Pasajeros Pasados (global y por intervalo), la Ocupación Media global de cada ascensor y su promedio, la Cantidad de Paradas global de cada ascensor y su suma, y finalmente los Largos de Cola Medios y Máximos globales de cada piso con su promedio y su máximo respectivamente.

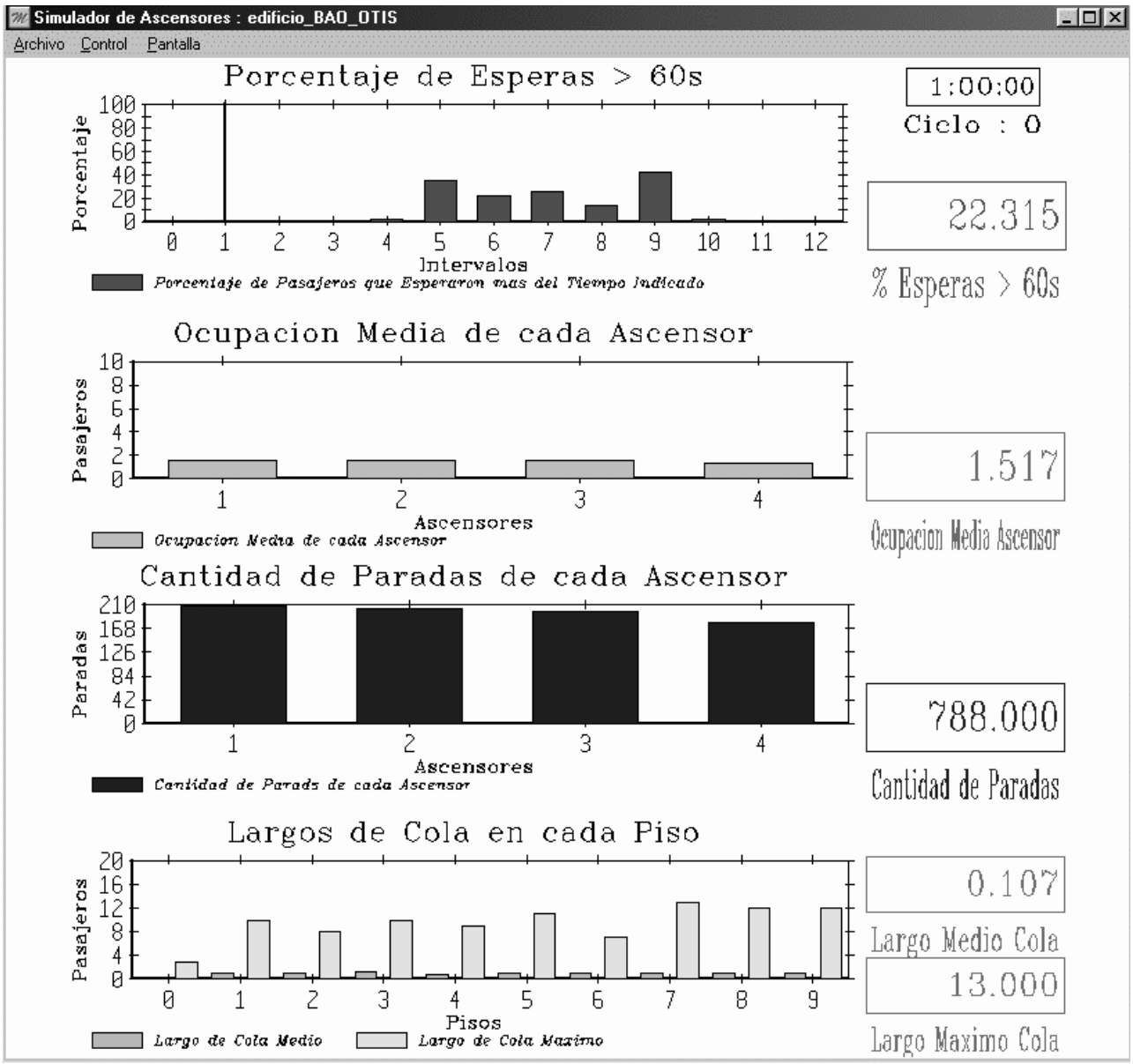

Simulador de Ascensores : edificio_BAO_OTIS
Archivo Control Pantalla
Porcentaje de Esperas > 60s
Porcentaje
Intervalos
Porcentaje de Pasajeros que Esperaron mas del Tiempo Indicado
1:00:00
Ciclo : 0
22.315
% Esperas > 60s
Ocupacion Media de cada Ascensor
Pasajeros
Ascensores
Ocupacion Media de cada Ascensor
1.517
Ocupacion Media Ascensor
Cantidad de Paradas de cada Ascensor
Paradas
Ascensores
Cantidad de Parads de cada Ascensor
788.000
Cantidad de Paradas
Largos de Cola en cada Piso
Pasajeros
Pisos
Largo de Cola Medio
Largo de Cola Maximo
0.107
Largo Medio Cola
13.000
Largo Maximo Cola

Figura 6.2.4.2

6) ANÁLISIS DE RESULTADOS Y VALIDACIÓN

6.1) Caso de Estudio

Nuestro caso de estudio básico es una recreación del utilizado por Bao et al en [15] (nos referiremos a él como Edificio Bao). Dicho caso de estudio les fue provisto por la compañía OTIS, por lo que podemos suponer que reflejan apropiadamente alguna realidad, y cotejando nuestros resultados con los de ellos podremos obtener una cierta validación de nuestro simulador.

Las características de nuestro caso de estudio son entonces las siguientes:

- 10 pisos (del 0 al 9) de 1.45 unidades de alto.
- 4 ascensores idénticos (del 1 al 4) con:
  - capacidad para 20 personas.
  - velocidad 1 unidad por segundo.
  - tiempo de arranque y parada 3.6 segundos.
- El tiempo de subida de los pasajeros sigue una distribución de Erlang de media 1 segundo y varianza 0.05.
- La llegada de pasajeros a los pisos sigue una distribución de Poisson que varía en un ciclo de 60 minutos, con las siguientes características:
  - En la planta baja, las llegadas son constantes a una tasa de 10 pasajeros cada 5 minutos (o sea, equivalente a distribución exponencial de frecuencia 30) destinados uniformemente a los otros 9 pisos.
  - En los pisos 1 a 9, las tasas de llegada de pasajeros hacia la planta baj a varían cada 5 minutos según especificamos más abajo. Además de estos, los pasajeros hacia pisos que no sean la planta baja llegan con una tasa del 10% las indicadas.

Intervalo(minutos)	Pasajeros/5minutos	Frecuencia(+10%)
0-5	1	272.727
5-10	2	136.364
10-15	4	68.182
15-20	4	68.182
20-25	18	15.152
25-30	12	22.727
30-35	8	34.091
35-40	7	38.961
40-45	18	15.152
45-50	5	54.545
50-55	3	90.909
55-60	2	136.364

Cabe hacer notar que estas llegadas de pasajeros configuran un caso de pico de bajada.

6.2) Validación del Simulador

Como ya mencionamos, implementamos cuatro schedulers, en versiones con y sin Balking y Reneging: IQS, IFC, HUFF Básico y HUFF. Para validar el simulador SSA, analizaremos los resultados obtenidos con las versiones normales y los compararemos con los resultados de [15].

SSA - Edificio Bao - 940 pasajeros/hora – 1 corrida de 4 ciclos						
	Espera	Espera Cuadrada	Tiempo Sistema	Tiempo Sist Cuadrado	Espera Máxima	% espera > 60 segundos
IQS con CdeG	34.374057	2228.784494	57.598833	4734.186754	183.423684	20.430966
IFC sin CdeG	48.240864	3776.216671	92.702002	10928.445834	277.368501	33.732376
HUFF Básico	24.534580	1006.434396	59.182574	4609.296454	168.581047	7.049747
HUFF	23.081364	916.892106	58.513580	4548.278794	134.534079	5.719606

En esta tabla vemos los resultados globales de nuestras corridas. Observamos que los resultados en general son lo que se esperaría, los algoritmos más modernos tienen mejor performance que los más viejos. La excepción es el IFC, que al no tener ningún control de grupos sufre seriamente de aglutinamiento y es notablemente peor que el IQS, en el que por definición nunca hay más de un ascensor atendiendo el mismo piso.
Asimismo, observamos que el IQS tiene una pequeña ventaja respecto a los HUFFs en cuanto a los tiempos de sistema, dado que al atender inmediatamente los pedidos internos reduce el tiempo de viaje, sobre todo en un caso de pico de bajada como este. Sin embargo, los tiempos de espera y la varianza de esos tiempos de espera (indicada por la sum a de sus cuadrados) es mucho peor que en cualquiera de los HUFFs.
Otro detalle que merece la pena mencionar es que el tiempo requerido por el simulador para completar cada ciclo de una hora es del orden de los 6 minutos para cualquiera de los algoritmos en un equipo Pentium III de 500 Mhz. Esta velocidad parece demasiado lenta para la capacidad del equipo y la complejidad de los algoritmos, lo que nos lleva a pensar que es una limitación del ModSim III , o por lo menos de su versión académica .

[15] – Edificio Bao - Promedios de 30 corridas de 1 ciclo - 895 pasajeros/hora					
	Espera	Espera Cuadrada	Tiempo Sistema	Espera Máxima	% espera > 60 segundos
HUFF Básico	22.0	756	51.1	91.1	1.99
HUFF	19.6	608	50.5	87.3	1.56

SSA – Edificio Bao - Promedios de 4 corridas de 1 ciclo - 944 pasajeros/hora					
	Espera	Espera Cuadrada	Tiempo Sistema	Espera Máxima	% espera > 60 segundos
HUFF Básico	25.208384	1077.2223	59.068963	130.04650	7.6623023
HUFF	22.284354	853.19706	57.131461	111.16051	5.1376863

En estas tablas comparamos los resultados de los algoritmos HUFFs obtenidos por Bao et al con los que nosotros obtuvimos. Vemos que si bien los valores de las esperas medias son muy compatibles,

nuestros resultados arrojan una mayor varianza de estos tiempos. Esto afecta a los demás valores, haciéndolo que aumenten las esperas máximas y los porcentajes de pasados, más allá de lo que en un primer momento se podría esperar por la mayor cantidad de pasajeros procesados. Sin embargo, esta diferencia en la cantidad de pasajeros surge por el porcentaje del 10% de tráfico entre pisos superiores, que en el caso de Bao et al, variaba de forma no especificada entre 0% y 10%. Como ambos algoritmos están diseñados para optimizar la bajada desde pisos superiores a la planta baja, en razonable que estos 50 pasajeros de más tengan un peso bastante significativo en la performance de los algoritmos.
Además, la implementación de los algoritmos no es exactamente la misma, dada la manera algo informal de especificarlos usada por Bao et al. Es lógico que esto también provoque algunas diferencias.
Por otro lado, lo que se mantiene uniformemente similar son las relaciones entre los resultados de ambos algoritmos. El HUFF básico es consistentemente peor que el HUFF en ambos casos, y lo es en proporciones similares.

SSA - Edificio Bao con 0% de trafico entre pisos - Promedios de 4 corridas de 1 ciclo - 863 pasajeros/hora					
	Espera	Espera Cuadrada	Tiempo Sistema	Espera Máxima	% espera > 60 segundos
HUFF Básico	19.462888	584.11850	50.641203	72.038385	0.8606615
HUFF	19.767700	638.63308	49.397639	84.470333	1.5603820

Si repetimos nuestras pruebas suprimiendo el 10% de tráfico entre los pisos superiores, obtenemos una notoria mejoría en los resultados. Para el caso del HUFF, sus cifras se vuelven equivalentes a las de Bao et al, a pesar que ahora hay unos 30 pasajeros menos en el sistema. Por su parte, el HUFF Básico también ha dado mejores resultados que en la prueba anterior, aún comparado con los de Bao et al., pero sorprendentemente la mejoría es tal que arroja aún mejores resultados que el HUFF. Esto no concuerda con las conclusiones de [15], que indican que el HUFF es mejor que el HUFF Básico en toda circunstancia, aún al reducir la cantidad de pasajeros.
Debemos concluir que esto es causado por las ya mencionadas diferencias en el algoritmo, las cuales están afectando más al HUFF Básico que al HUFF, haciéndolo más eficiente en este caso.
Dado que la única diferencia con los parámetros de la prueba anterior es la falta de un pequeño volumen de tráfico entre pisos pero su efecto en los resultados es bastante significativo, es razonable sospechar que es la ausencia de pasajeros que *suben* la principal responsable de los cambios en los resultados (Recordemos que el algoritmo está optimizado para tráfico de bajada).
Si estamos en lo cierto, se debería observar el mismo fenómeno en un edificio con sólo tráfico de bajada:

[15] – Edificio Bao Solo Bajada - Promedios de 30 corridas de 1 ciclo - 760 pasajeros/hora					
	Espera	Espera Cuadrada	Tiempo Sistema	Espera Máxima	% espera > 60 segundos
HUFF Básico	19.85	580	47.2	66.8	0.76
HUFF	16.76	396	48.6	61.7	0.16

SSA - Edificio Bao Solo Bajada - Promedios de 4 corridas de 1 ciclo - 723 pasajeros/hora					
	Espera	Espera Cuadrada	Tiempo Sistema	Espera Máxima	% espera > 60 segundos
HUFF Básico	19.363261	516.28649	49.021054	57.847481	0.1016260
HUFF	19.188333	529.48473	49.516417	65.476164	0.4962987

Efectivamente, podemos ver que según [15], HUFF Básico arroja resultados bastante peores que el HUFF, pero en nuestro sistema los resultados son equivalentes o mejores. Concluímos entonces que efectivamente, las diferencias de implementación del HUFF Básico están causando diferencias particularmente notables ante la ausencia de tráfico de subida.

Por su parte, el comportamiento de nuestro HUFF es comparable con el de [15], pues si bien nuestros resultados son sistemáticamente peores, esta diferencia se mantiene uniforme al comparar corridas con diferentes parámetros de entrada.

En conclusión, nuestros análisis soportan la tesis que el simulador SSA se comporta de manera consistente y ajustada a la realidad que modela, y puede reproducir los resultados de otros simuladores si se dispone de los mismos algoritmos y entradas de datos.

6.3) Análisis de Balking y Reneging

Una de las particularidades del SSA es el soporte de Balking y Reneging. Estos fenómenos no suelen ser contemplados en simuladores de ascensores, a pesar de que son relativamente simples de implementar y de que en la vida real indiscutiblemente ocurren. Cabe preguntarse entonces si esta omisión puede afectar significativamente los resultados de un análisis, o si por el contrario estos dos fenómenos son, o estadísticamente irrelevantes, o fáciles de compensar.

Para investigar esta cuestión diseñamos el siguiente experimento, que aplicaremos tanto al Balking como al Reneging.

- Ejecutamos una serie de corridas sobre nuestro edificio de prueba, una con el algoritmo HUFF común y otra con su versión con Balking o Reneging.
- Analizaremos las diferencias en los resultados, y si son significativas, intentaremos ajustar la entrada del simulador para que el algoritmo HUFF dé resultados más parecidos. Esto lo haremos con la única herramienta disponible, modificando las tasas de llegada de cada intervalo para que reflejen la cantidad de pasajeros que abandonaron el sistema por Balking o Reneging.
- Los resultados de las corridas con esa entrada modificada nos mostrarán si estos fenómenos pueden ser compensados en un simulador corriente.

Cabe notar que las modificaciones a la entrada se harán con el mayor nivel de detalle posible, lo que nos debería dar la mejor aproximación posible. Sin embargo, también son trabajosas de efectuar y requieren de información muy detallada sobre el comportamiento de los pasajeros, que en un caso real probablemente no se encontrara disponible.

6.3.1) Balking

Efectuamos cuatro corridas sobre nuestro edificio con HUFF y con HUFF con Balking con un largo de cola crítico de tres pasajeros. Los resultados comparativos fueron:

SSA - Edificio Bao - Promedios de 4 corridas de 1 ciclo - 944 pasajeros/hora					
	Espera	Espera Cuadrada	Tiempo Sistema	Espera Máxima	% espera > 60 segundos
HUFF	22.284354	853.19706	57.131461	111.16051	5.1376863
HUFF Balk 3	20.857477	726.39018	50.141608	141.90275	2.7126063

Podemos ver que hay diferencias significativas en los resultados, como era de esperarse. Después de todo, a causa del Balking literalmente no entraron nuevos pasajeros al sistema en los pisos en que hubiera tres pasajeros esperando el ascensor.

Veamos ahora en qué intervalos y pisos hubieron pasajeros que se retiraron por Balking, y cuantos. Luego, ajustemos las frecuencias de llegada en esos pisos e intervalos para que se generen esas cantidades menos de pasajeros, y ejecutemos de vuelta las corridas.

SSA - Edificio Bao – HUFF con Balking a largo de cola 3 – promedios en 4 corridas de 1 ciclo de la cantidad de pasajeros que balkearon											
Intervalo	Piso 0	Piso1	Piso 2	Piso 3	Piso 4	Piso 5	Piso 6	Piso 7	Piso 8	Piso 9	TOTAL
01) 00-05	0	0	0	0	0	0	0	0	0	0	**0**
02) 05-10	0	0	0	0	0	0	0	0	0	0	**0**
03) 10-15	0.25	0	0	0	0	0	0	0	0	0	**0.25**
04) 15-20	0	0	0	0	0	0	0	0	0	0	**0**
05) 20-25	0	4.50	4.00	8.00	4.25	2.75	2.75	3.50	3.00	2.75	**35.50**
06) 25-30	0.50	1.75	2.75	1.50	1.75	0.75	1.25	1.00	0.25	2.50	**14.00**
07) 30-35	0	0.25	0	0	0.25	0.25	0	0.75	0.75	0.50	**2.75**
08) 35-40	0.25	0.25	0	0	0	0	0	0	0	0.75	**1.25**
09) 40-45	0.75	5.25	4.00	5.75	6.50	6.00	3.25	4.25	6.25	7.00	**49.00**
10) 45-50	0.25	0.25	0	0	0.25	0	0.50	0	1.00	0.25	**2.50**
11) 50-55	0	0	0	0	0	0	0	0	0	0	**0**
12) 55-60	0	0	0	0	0	0	0	0	0	0	**0**
TOTAL	**2.00**	**12.25**	**10.75**	**15.25**	**13.00**	**9.75**	**7.75**	**9.50**	**11.25**	**13.75**	**105.25**

SSA - Edificio Bao – Nuevas frecuencias de llegada para cada piso/intervalo ajustadas según la tabla anterior de Balking a largo de cola 3										
Intervalo	Piso 0	Piso1	Piso 2	Piso 3	Piso 4	Piso 5	Piso 6	Piso 7	Piso 8	Piso 9
00-05	30.000	272.727	272.727	272.727	272.727	272.727	272.727	272.727	272.727	272.727
05-10	30.000	136.364	136.364	136.364	136.364	136.364	136.364	136.364	136.364	136.364
10-15	30.769	68.182	68.182	68.182	68.182	68.182	68.182	68.182	68.182	68.182
15-20	30.000	68.182	68.182	68.182	68.182	68.182	68.182	68.182	68.182	68.182
20-25	30.000	19.608	18.987	25.424	19.293	17.595	17.595	18.405	17.857	17.595
25-30	31.579	26.201	28.708	25.641	26.201	24.096	25.105	24.590	23.166	28.037
30-35	30.000	35.088	34.091	34.091	35.088	35.088	34.091	37.267	37.267	36.144
35-40	30.769	40.268	38.961	38.961	38.961	38.961	38.961	38.961	38.961	43.165
40-45	32.432	20.619	18.987	21.352	22.556	21.739	18.127	19.293	22.140	23.438
45-50	30.769	57.143	54.545	54.545	57.143	54.545	60.000	54.545	66.667	57.143
50-55	30.000	90.909	90.909	90.909	90.909	90.909	90.909	90.909	90.909	90.909
55-60	30.000	136.364	136.364	136.364	136.364	136.364	136.364	136.364	136.364	136.364

SSA - Edificio Bao ajustado según tabla anterior - Promedios de 4 corridas de 1 ciclo - 826 pasajeros/hora					
	Espera	Espera Cuadrada	Tiempo Sistema	Espera Máxima	% espera > 60 segundos
HUFF	20.132853	777.18144	49.715021	170.17127	2.6900588
% Diferencia con HUFF Balk3 en el Ed. Orig.	-3.474	+6.992	-0.851	+19.921	-0.831

Vemos que los resultados varían según la medida. El tiempo del sistema es prácticamente idéntico, así como el porcentaje de pasados de 60 segundos, mientras que la espera máxima fue muy diferente.

Por otro lado, la espera media tuvo una variación pequeña aunque no insignificante. Algo mayor fue el cambio de la espera cuadrada media, lo que indica que las esperas medias fueron menos uniformes.

En general, podemos decir que excepto para la espera máxima, las diferencias son lo bastante pequeñas como para poder utilizar los resultados. No obstante, esta aproximación requirió un ajuste laborioso de los datos de entrada, y un conocimiento del comportamiento de los pasajeros mucho más detallado y difícil de obtener que un valor de largo de cola límite para el Balking.

6.3.2) Reneging

Efectuamos las mismas corridas con el algoritmo HUFF con Reneging con tiempo de espera límite de 60 segundos. Los resultados son los siguientes:

SSA - Edificio Bao - Promedios de 4 corridas de 1 ciclo - 944 pasajeros/hora					
	Espera	Espera Cuadrada	Tiempo Sistema	Espera Máxima	% espera > 60 segundos
HUFF	22.284354	853.19706	57.131461	111.16051	5.1376863
HUFF Ren 60	19.544707	599.45751	52.870385	59.806401	0

Podemos ver que las diferencias son bastante mayores que para el caso del Balking, aún descontando la espera máxima, que obviamente está acotada a 60 segundos, y el porcentaje de pasados de 60 segundos, que necesariamente vale 0.

Si repetimos los análisis que hicimos en el caso anterior, obtenemos estos resultados:

SSA - Edificio Bao – HUFF con Reneging a 60 segundos – promedios en 4 corridas de 1 ciclo de la cantidad de pasajeros que renegaron											
Intervalo	Piso 0	Piso1	Piso 2	Piso 3	Piso 4	Piso 5	Piso 6	Piso 7	Piso 8	Piso 9	TOTAL
01) 00-05	0	0	0	0	0	0	0	0	0	0	**0**
02) 05-10	0	0	0	0	0	0	0	0	0	0	**0**
03) 10-15	0	0	0	0	0	0	0	0	0	0	**0**
04) 15-20	0	0	0	0	0	0	0	0	0	0	**0**
05) 20-25	0.25	3.25	1.25	1.50	1.25	0.25	0.50	0.75	0	0.50	**9.50**
06) 25-30	0.50	0.25	0	1.00	0.75	0.50	1.00	0	0	0.50	**4.50**
07) 30-35	0	0.25	0	0	0.50	0.25	0	0	0.25	0	**1.25**
08) 35-40	0.25	0	0	0	0	0	0.25	0.50	0.75	0	**1.75**
09) 40-45	1,25	1.50	2.50	1.00	1.00	1.50	2.00	1.25	2.00	3.00	**17.00**
10) 45-50	0	0	0	0	0	0	0	0	0	0	**0**
11) 50-55	0	0	0	0	0	0	0	0	0	0	**0**
12) 55-60	0	0	0	0	0	0	0	0	0	0	**0**
TOTAL	**2.25**	**5.25**	**3.75**	**3.50**	**3.50**	**2.50**	**3.75**	**2.50**	**3.00**	**4.00**	**34.00**

SSA - Edificio Bao – Nuevas frecuencias de llegada para cada piso/intervalo ajustadas según la tabla anterior de Reneging a espera 60s										
Intervalo	Piso 0	Piso1	Piso 2	Piso 3	Piso 4	Piso 5	Piso 6	Piso 7	Piso 8	Piso 9
00-05	30.000	272.727	272.727	272.727	272.727	272.727	272.727	272.727	272.727	272.727
05-10	30.000	136.364	136.364	136.364	136.364	136.364	136.364	136.364	136.364	136.364
10-15	30.000	68.182	68.182	68.182	68.182	68.182	68.182	68.182	68.182	68.182
15-20	30.000	68.182	68.182	68.182	68.182	68.182	68.182	68.182	68.182	68.182
20-25	30.769	18.127	16.173	16.393	16.173	15.345	15.544	15.748	15.152	15.544
25-30	31.579	23.166	22.727	24.590	24.096	23.622	24.590	22.727	22.727	23.622
30-35	30.000	35.088	34.091	34.091	36.145	35.088	34.091	34.091	35.088	34.091
35-40	30.769	38.961	38.961	38.961	38.961	38.961	40.268	41.667	43.165	38.961
40-45	34.286	16.393	17.341	15.957	15.957	16.393	16.854	16.173	16.854	17.857
45-50	30.000	54.545	54.545	54.545	54.545	54.545	54.545	54.545	54.545	54.545
50-55	30.000	90.909	90.909	90.909	90.909	90.909	90.909	90.909	90.909	90.909
55-60	30.000	136.364	136.364	136.364	136.364	136.364	136.364	136.364	136.364	136.364

SSA - Edificio Bao ajustado según tabla anterior - Promedios de 4 corridas de 1 ciclo - 915 pasajeros/hora					
	Espera	Espera Cuadrada	Tiempo Sistema	Espera Máxima	% espera > 60 segundos
HUFF	20.797562	735.99944	53.551313	129.27691	3.6591595
% Diferencia con HUFF Reneg60 en el Ed. Orig.	+6.410	+22.778	+1.288	+116.159	N/A

En este caso, vemos que la única medida que se aproxima al objetivo es el tiempo de sistema medio. La espera máxima y el porcentaje de pasados obviamente no se ciñen al límite de 60 segundos, mientras que por su parte la espera media, y sobre todo la espera media cuadrada tienen variaciones muy superiores a las obtenidas en el caso del Balking, demasiado como para poder utilizar estos resultados con algún grado de confianza.
Esto no es extraño, ya que en el Reneging el pasajero existe dentro del sistema y lo deja después del tiempo límite, afectando las estadísticas, a diferencia del Balking, cuyo comportamiento es equiparable a una reducción muy específica de las llegadas.

6.3.3) Conclusiones del Análisis

Hemos visto que los efectos del Balking se pueden emular en un simulador convencional con trabajo e información, pero el Reneging no es tan fácil de emular. Esto también se aplica forzosamente a los casos en que se contemplan ambos fenómenos simultáneamente.

Dado que, como vimos, los efectos de estos fenómenos pueden ser significativos (lo que quedó demostrado por nuestro ejemplo, en el que efectivamente lo son), podemos concluir que es deseable para un simulador de ascensores el incluir la posibilidad de manejarlos direct amente.

7) CONCLUSIONES Y TRABAJO A FUTURO

Nuestro análisis de la investigación matemática sobre el problema del control de grupos de ascensores arrojó como resultado que el campo está aún en su infancia. Si bien el problema del ruteo de un único ascensor ha sido formalizado y estudiado, conociéndose numerosos algoritmos de eficiencia y efectividad demostradas para su solución, no pudimos encontrar ninguna referencia al caso del ruteo de múltiples ascensores. Para paliar esta laguna, decidimos definir formalmente un problema de ruteo de múltiples ascensores y demostrar su NP-Completitud. Esperamos que esta definición pueda servir de estímulo y punto de partida para estudios más profundos sobre este problema.

Por el contrario, en el ámbito académico de la investigación operativa se encuentran numerosos estudios sobre el control de grupos de ascensores. Sin embargo, sus fundamentos son más empíricos que matemáticos. En efecto, hemos comprobado que en estos trabajos la eficiencia es función del hardware que se utilice, y el grado de eficacia de una estrategia de ruteo se comprueba comparando sus resultados promedio con los de otras estrategias conocidas. En el ámbito industrial también existen numerosos sistemas para el control de ascensores, pero desgraciadamente los detalles de los mismos son considerados propietarios de las empresas y rara vez son descriptos más allá de sus líneas generales. Esto dificulta seriamente el poder recopilar un estudio completo del estado del arte de este problema.

El uso de la simulación en este campo tiene una larga historia, y su aceptación como herramienta para la investigación está firmemente establecida. Las pruebas que efectuamos con el SSA nos llevan a concluir que es una herramienta efectiva para el análisis de los sist emas de ascensores, comparable a otros simuladores. Además, su capacidad de manejar Balking y Reneging es un recurso valioso para ese fin.
Sin embargo, hay mucho lugar para mejoras. Un requerimiento que se puso en claramente en evidencia luego del desarrollo fue la falta de un modo de operación batch. Otras funcionalidades que podrían ser deseables son una salida gráfica que muestre el movimiento de los ascensores durante la corrida, y la posibilidad de grabar a disco un registro de todos los movimientos de los ascensores y pasajeros, para su posterior análisis detallado.
De cualquier manera, los principales inconvenientes que encontramos (la velocidad de ejecución, dificultades de integración con C++, falta de rollback) surgen de que el SSA está escrito en un lenguaje efectivamente muerto como el ModSim III. El próximo paso debe ser migrarlo a algún lenguaje de simulación moderno, preferiblemente uno que soporte mejor integración con programas escritos en otros lenguajes (para poder interfacear por ejemplo con sistemas de redes neurales) y la capacidad de efectuar rollback (para poder implementar con facilidad algoritmos con estimación futura).

Otro trabajo a continuar en el futuro es dar difusión al formato de entrada de datos del SSA y proponer la idea de usarlo como base de un formato universal para la entrada de simuladores de ascensores. Los comentarios y requerimientos que se reciban serán invaluables, ya sea para continuar el desarrollo en esa dirección o por el contrario para decidir que el proyecto es impracticable.

BIBLIOGRAFÍA

[1] El Libro del Transporte Vertical - Antonio Miravete, Emilio Larrodé - Servicio de Publicaciones, Centro Politécnico Superior, Universidad de Zaragoza - 1996

[2] Planning and Control Models for Elevators in High -Rise Buildings - Marja-Liisa Siikonen - Helsinki University of Technology, Systems Analysis Laboratory Research Reports, A68 - October 1997
<http://citeseer.nj.nec.com/siikonen97planning.html>

[3] An Overall Review of Advanced Elevator Technologies - Dr. Albert T.P. So, S.K. Liu - ELEVATOR WORLD Magazine VOL XLIV, NO. 6 pag. 96-101 - June 1996
http://www.elevator-world.com/magazine/archive01/9606-001.htm>

[4] Series GPS-III Passenger Elevators - Mitsubishi Electric Corporation (Pagina Web visitada 15/06/2002)
<http://www.mitsubishi-elevator.com/design/ele/gps3/gps3_art.htm>

[5] Improving Elevator Performance Using Reinforcement Learning - Robert H. Crites, Andrew G. Barto - Advances in Neural Information Processing Systems VOL 8 pag 1017-1023, The MIT Press – 1996
<http://citeseer.nj.nec.com/crites96improving.html>

[6] Online Optimization -- Competitive Analysis and Beyond. - S.O. Krumke. - Habilitation Thesis, Technical University of Berlin, 2001 - 2001
<http://www.zib.de/krumke/Postscript/habil.ps.gz>
<ftp://ftp.zib.de/pub/zib-publications/reports/ZR-02-25.ps.Z>

[7] Routing a Vehicle of Capacity Greater Than One - D. J. Guan - DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science 81 (1998), no. 1, pag 41-57 – 1998
<http://citeseer.nj.nec.com/guan96routing.html>

[8] The Finite Capacity Dial-A-Ride Problem - Moses Charikar, Balaji Raghavachari - IEEE Symposium on Foundations of Computer Science, pag 458-467 – 1998
<http://citeseer.nj.nec.com/493621.html>

[9] An Approximation Algorithm for the Non-Preemptive Capacitated Dial-a-Ride Problem - Sven O. Krumke, Jörg Rambau, Steffen Weider - Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Report No. 00-53 – 2000
<http://citeseer.nj.nec.com/krumke00approximation.html>

[10] The Online Transportation Problem: Competitive Scheduling of Elevators - Norbert Ascheuer, Sven O. Krumke, Jörg Rambau - Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Report No. 98-34 (Appeared as: Online Dial-a-Ride Problems: in: Minimizing the Completion Time. Proceeding of the 17th International Symposium on Theoretical Aspects of Computer Science, Vol 1770 of Lecture Notes in Computer Science, Springer 2000, 639-650) – 1998 <http://citeseer.nj.nec.com/106625.html>

[11] Real Time Optimization of Elevator Systems - Sven O. Krumke, Jörg Rambau (Página Web visitada 30/06/2002)
<http://www.zib.de/projects/production-planning/elevator/index.en.html>

[12] Optimal Dispatching Control for Elevator Systems During Uppeak Traffic - David L. Pepyne and Christos G. Cassandras - IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, VOL. 5, NO. 6, NOVEMBER 1997 pag 629-643 - 1997

[13] Dynamic Zoning in Elevator Traffic Control - W.L. Chan, A.T.P. So, K.C. Lam - ELEVATOR WORLD Magazine VOL XLV, NO. 3 pag 136-139 – March 1997
<http://www.elevator-world.com/magazine/archive01/9703-002.htm>

[14] Elevator Group Control Using Multiple Reinforcement Learning Agents - Robert H. Crites, Andrew G. Barto - Machine Learning VOL 33 no 2-3 pag 235-262 - 1998
<http://citeseer.nj.nec.com/crites98elevator.html>

[15] Elevators Dispatchers for Down Peak Traffic – G. Bao, C. G. Cassandras, T. E. Djaferis, A. D. Gandhi, and D. P. Looze – Technical Report, ECE Department, University of Massachusetts, Amherst, MA - 1994

[16] Design and Implementation of an Adaptive Dispatching Controller for Elevator Systems During Uppeak Traffic - David L. Pepyne and Christos G. Cassandras - IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, VOL. 6, NO. 5, SEPTEMBER 1998 pag 635-650 - 1998
<http://vita.bu.edu/cgc/Published/TCSTprint0998.pdf>

[17] Genetic Algorithms and Discrete Event Systems: An Application - Ricardo R. Gudwin, Fernando A. C. Gomide - Proceedings of The First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, 26 de Junho a 2 de Julho de 1994, Orlando, Florida, USA, pag 742-745 - 1994
<http://citeseer.nj.nec.com/443900.html>

[18] Logical Control of an Elevator with Defeasible Logic - Michael A. Covington - Technical Report, Artificial Intelligence Center, University of Georgia - 16/02/1999
<http://citeseer.nj.nec.com/474405.html>

[19] A Fuzzy Elevator Group Controller with Linear Context Adaptation - Ricardo Gudwin, Fernando Gomide, Márcio Andrade Netto - Proceedings of FUZZ-IEEE98, WCCI'98 - IEEE World Congress on Computational Intelligence, 4-9 May 1998, Anchorage, Alaska, USA, pag. 481-486 - 1998
<http://citeseer.nj.nec.com/241135.html>

[20] NN Elevator Group Control Method - Wan Jianru, Liu Chunjiang and Liu Hongchi - Tianjin University, Tianjin, China - ELEVATOR WORLD Magazine VOL L, NO 2 pag 148-154 - February 2002
<http://www.elevator-world.com/magazine/pdf/0202-003.pdf>

[21] Elevator Control as a Planning Problem - Jana Koehler, Kilian Schuster - Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS00) . pag 331-338 , AAI Press, Menlo Park – 2000
<http://citeseer.nj.nec.com/koehler00elevator.html>

[22] Virtual Simulation Environments of Elevator Group Control Systems - Zong Qun, Xue Li-Hua and Wang Zhen-Shi - School of Electrical Automation and Power Engineering, Tianjin University - ELEVATOR WORLD Magazine VOL XLIX, NO. 12 pag 90-93 - December 2001
<http://www.elevator-world.com/magazine/pdf/0112-002.pdf>

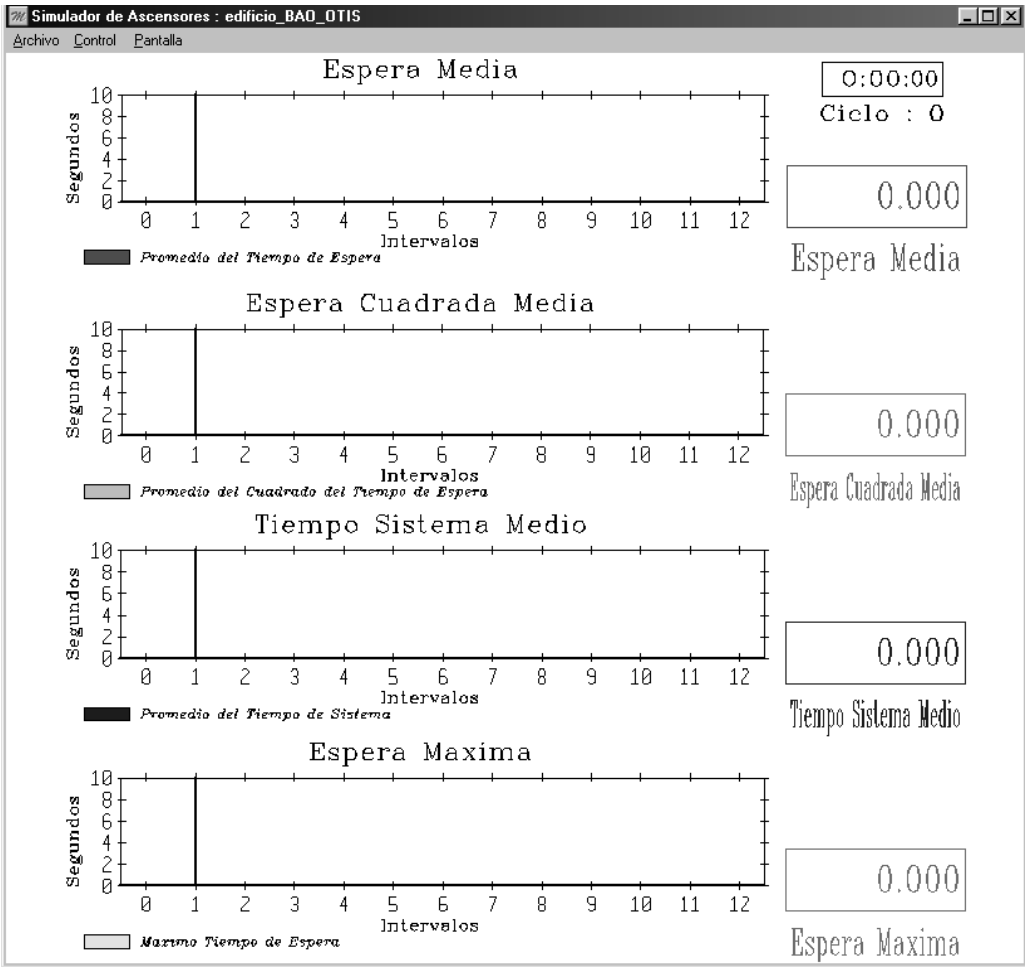
APENDICE A: Manual de Operación del SSA

A.1) Interfaz Gráfico

El control sobre el SSA se efectúa exclusivamente a través de un interfaz gráfico.
Una vez arrancado el programa *ascensor.exe*, se abre una ventana en blanco con las siguientes opciones de menú:

El submenú *Archivo* contiene los comandos de manejo de archivos y control básico del programa:

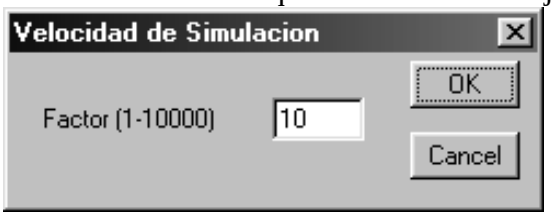
- *Abrir*: Abre un nuevo archivo de definición de edificio. (ver apéndice A)
  Al abrir un edificio nuevo se abre el diálogo de Control de Opciones (ver más abajo) que define las características de la corrida. Luego aparecen en la ventana los gráficos apropiados a las opciones elegidas.



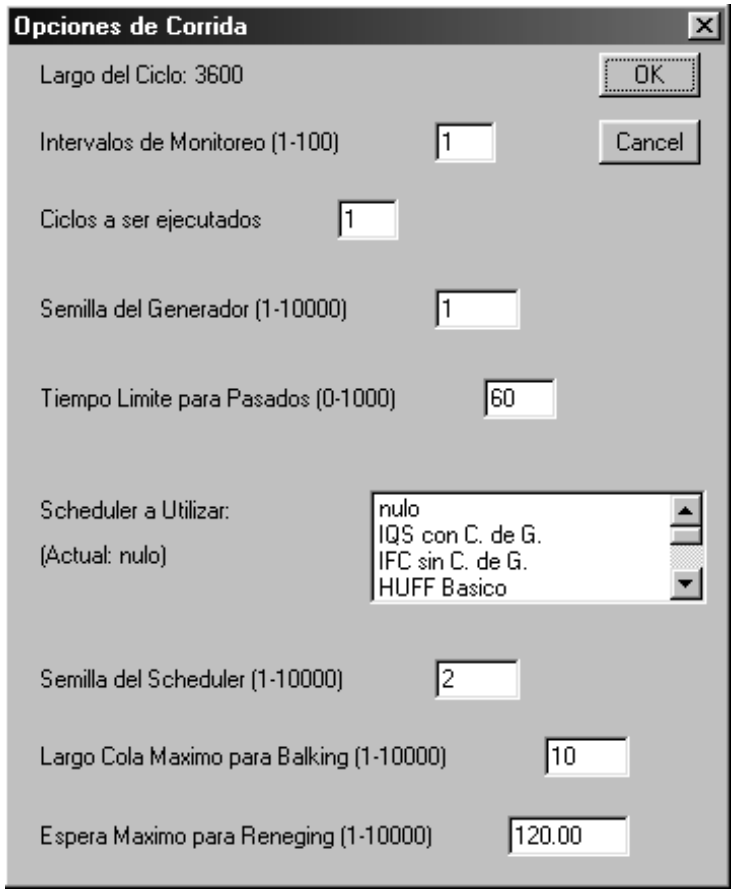
- *Grabar*: Graba el reporte de salida de la corrida que haya terminado. (ver apéndice C)
- *Imprimir*: Imprime la pantalla que se está viendo actualmente.
- *Cerrar*: Cierra los datos del edificio actualmente abierto.
- *Salir*: Sale del programa.

El submenú Control contiene los comandos de control de la corrida:

- *Comenzar*: Arranca una nueva corrida de simulación
- *Detener*: Interrumpe una corrida de simulación q ue esté en proceso
- *Resetear*: Descarta los datos
- *Velocidad*: Abre un diálogo que permite cambiar la velocidad a la que transcurre el tiempo dentro de la simulación. Mientras más alto el valor ingresado, más rápido progresa la simulación. Un valor de 1 indica que la simulación se ejecuta en tiempo real.



- *Opciones*: Abre un diálogo para modificar las opciones de la corrida. Este diálogo se abre también automáticamente cuando se abre un nuevo edificio.

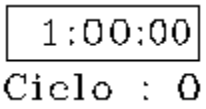


En este diálogo se reporta el largo total del ciclo definido por el edificio, y se pueden modificar los siguientes valores:

- *Intervalos de Monitoreo*: Es la cantidad de intervalos iguales en los que se subdivide el ciclo (y las gráficas) a los efectos de recolectar estadísticas.
- *Ciclos a ser ejecutados*: Es la cantidad de veces que se repetirá el ciclo definido por el edificio en esta corrida .
- *Semilla del Generador*: Es la semilla a usar en esta corrida para el generador de números aleatorios utilizado para manejar la llegada de pasajeros.
- *Tiempo Límite para Pasados*: Uno de los reportes del simulador es el porcentaje de pasajeros que esperaron más de un tiempo límite dado. Este valor especifica en segundos el tiempo límite a utilizar en esta corrida.
- *Scheduler a utilizar*: Es para elegir el algoritmo de rut eo a ser usado en esta corrida.
- *Semilla del Scheduler*: Es la semilla a usar en esta corrida para el generador de números aleatorios del scheduler (no tiene efecto si el algoritmo elegido no utiliza dicho generador)
- *Largo Cola Máximo para Balking*: Es el largo de cola crítico ante el cual un pasajero abandona el sistema (no tiene efecto si el algoritmo elegido no contempla Balking)
- *Espera Máxima para Reneging*: Es el tiempo de espera crítico ante el cual un pasajero abandona la cola de espera del ascensor (no tiene efecto si el algoritmo elegido no contempla Reneging)

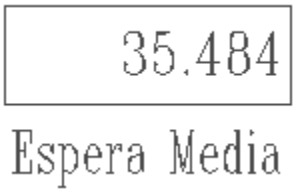
El submenú Pantalla contiene los comandos para definir lo que se ve en la pantalla:

- *Reloj*: Muestra y oculta el reloj de la simulación que se encuentra en el ángulo superior derecho:



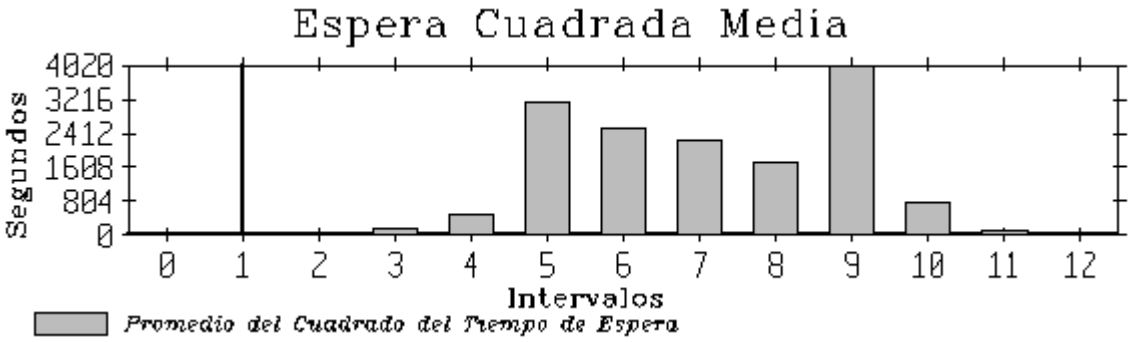
Este reloj muestra el tiempo de simulación en formato HH:MM:SS , y la cantidad de ciclos que han terminado.

- *Gráficas Tiempo*: Muestra y oculta las gráficas de las estadísticas que se recolectan en tiempo real durante la corrida. Estas son:
  - El promedio de los tiempos de espera de cada pasajero, tanto globales como dentro de cada intervalo

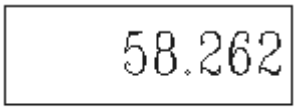


- El promedio de los cuadrados de los tiempos de espera de cada pasajero, tanto globales como dentro de cada intervalo

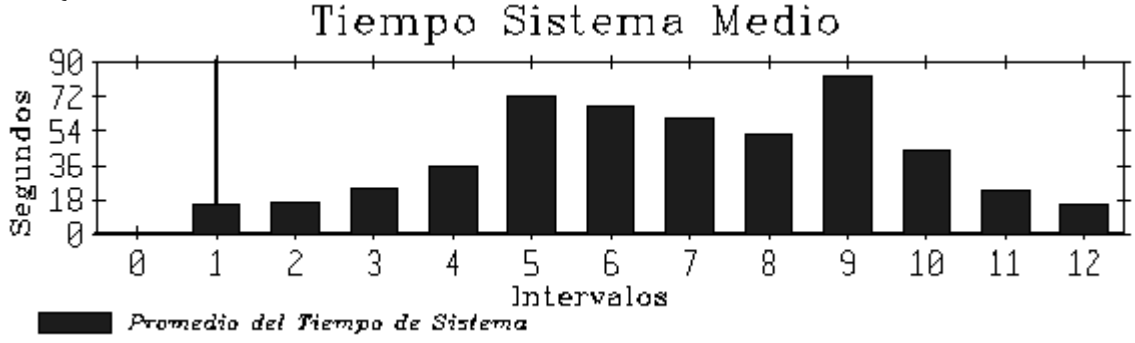
Espera Cuadrada Media



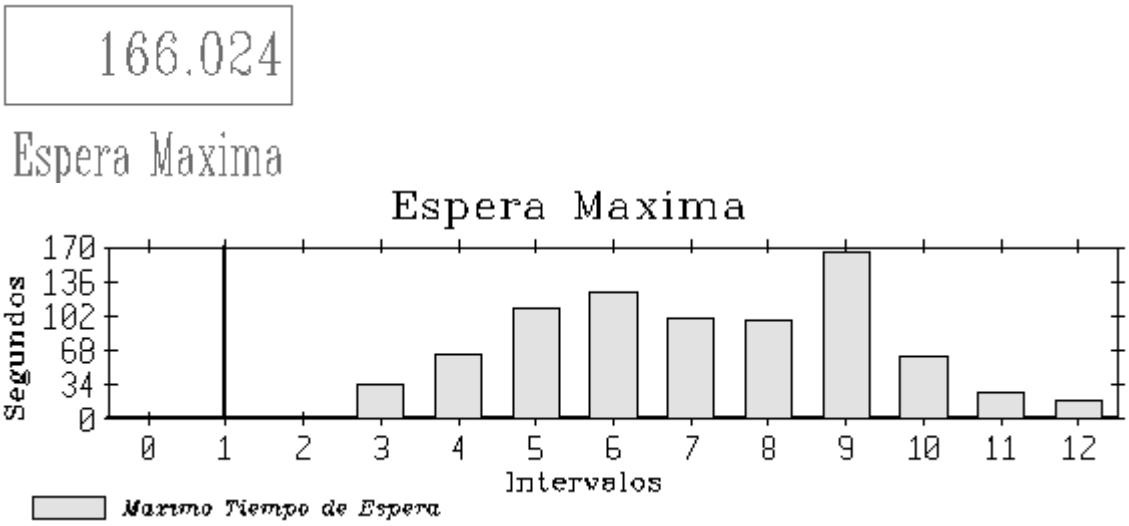
- El promedio de los tiempos de sistema totales de cada pasajero, tanto globales como para cada intervalo



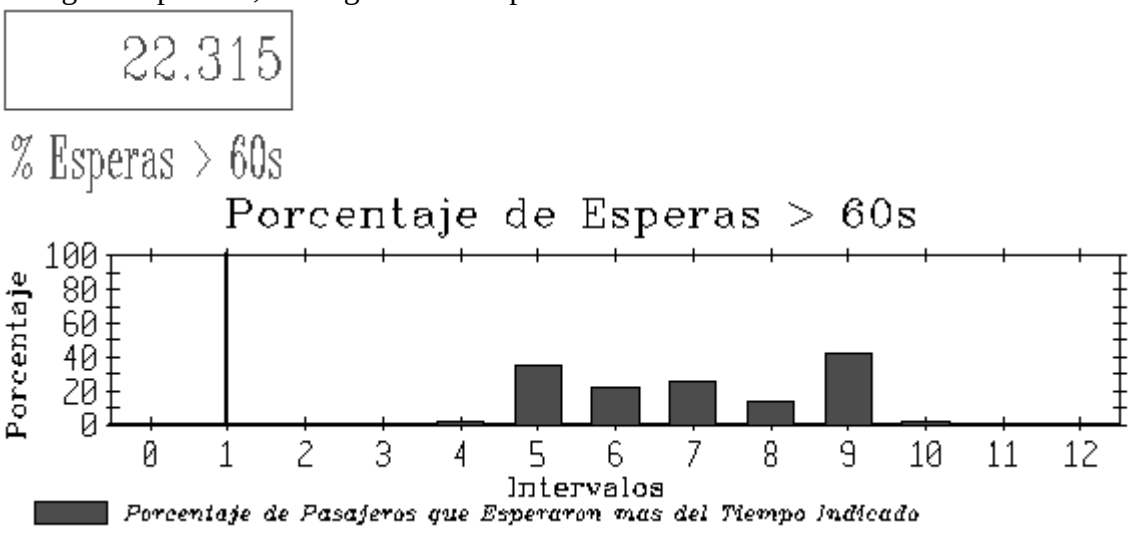
Tiempo Sistema Medio



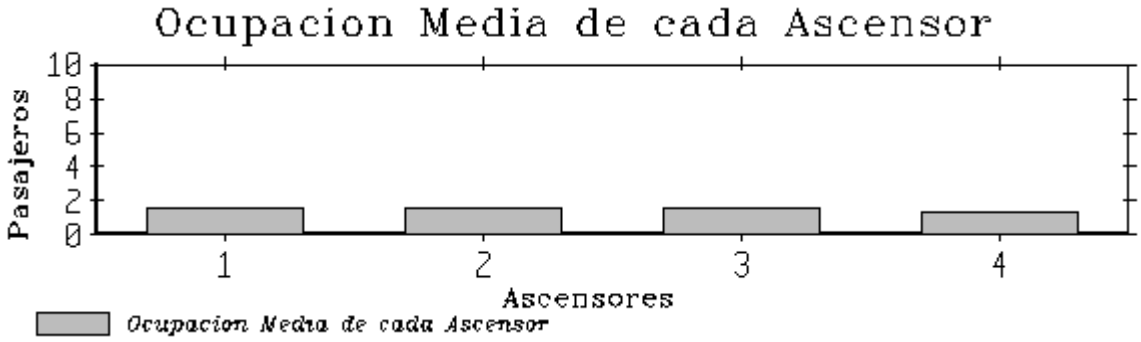
- La espera máxima que haya experimentado un pasajero, tanto global como para cada intervalo.



- *Gráficas Totales*: Muestra y oculta las gráficas de las estadísticas que se calculan al terminar una corrida. Estas son:
  - El porcentaje de pasajeros que esperaron más que el tiempo límite especificado en el diálogo de opciones, tanto global como para cada intervalo.



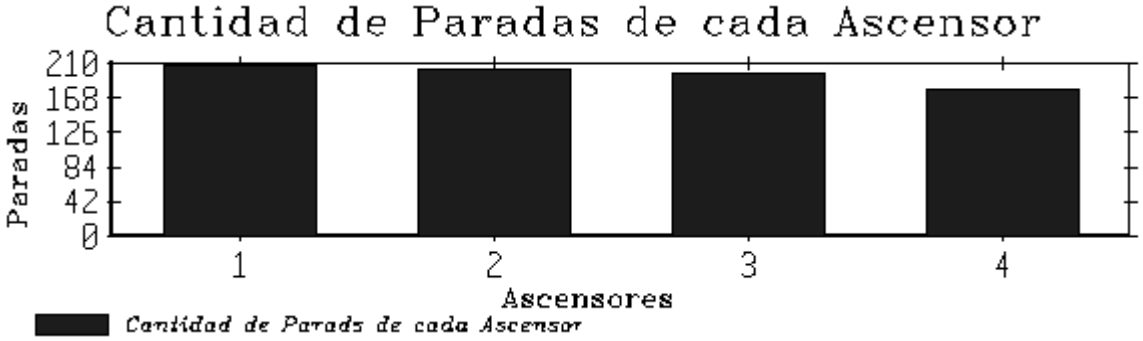
  - La ocupación media ponderada en el tiempo de los ascensores durante el total de la corrida, tanto conjunta como para cada ascensor individualmente.



- La cantidad de paradas de los ascensores durante el total de la corrida, tanto conjunta como para cada ascensor individualmente.

788.000

Cantidad de Paradas



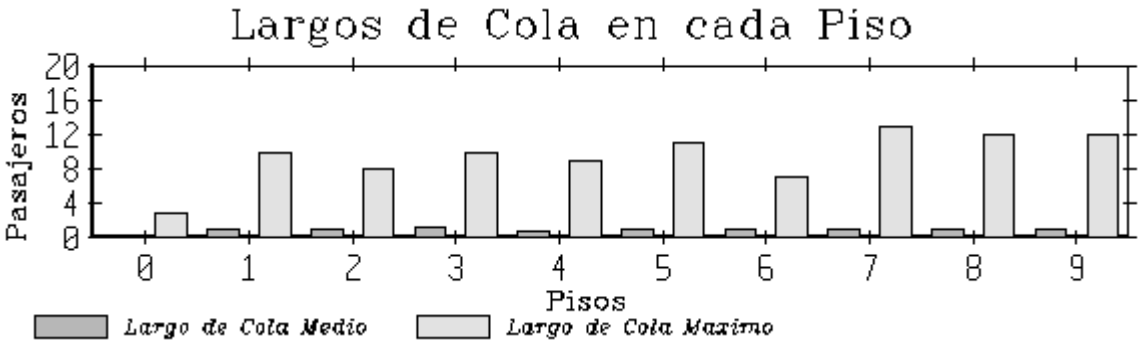
- Los largos de cola de espera de pasajeros medios ponderados en el tiempo durante el total de la corrida, tanto conjuntos como para cada piso individualmente, y sus máximos respectivos.

0.107

Largo Medio Cola

13.000

Largo Maximo Cola



A.2) Ejemplo de Utilización

Una sesión normal de uso del programa SSA tendría por lo tanto estos pasos:

1. Iniciar el programa.
2. Abrir un edificio.
3. Ajustar las opciones.
4. Ajustar la velocidad.
5. Comenzar la corrida.
6. Esperar que termine, o detenerla manualmente.
7. Ver (y opcionalmente, imprimir) ambas pantallas de gráficas.
8. Grabar el archivo de resultados de la corrida.
9. Si se desea efectuar otra corrida con este mismo edificio, resetear la corrida y volver al paso 3.
   Si se desea trabajar con otro edificio, cerrar este e ir al paso 2.
   Si se desea salir del programa, usar la opción de salir.

APENDICE B: Definición de un Nuevo Manejador de Ascensores

Para poder definir nuevos manejadores y algoritmos de ruteo en el SSA, es requisito indispensable conocer el lenguaje ModSimIII. Este lenguaje tiene una gramática similar a la del PASCAL, por lo que su aprendizaje no debería resultar muy complicado.

Una vez que se conoce el lenguaje, el procedimiento para crear y agregar un nuevo algoritmo es relativamente simple.

Los cambios se efectúan dentro del módulo *Scheduler*, compuesto por los archivos Dscheduler.mod (archivo de definiciones) e Ischeduler.mod (archivo de implementación).

B.1) Procedimiento para Crearlo

En este módulo los manejadores están definidos como objetos descendientes del tipo SchedulerObj, el cual define los recursos básicos y el interface para controlar los as censores.

SchedulerObj

```
nombre:STRING;              {nombre del manejador}
generador:RandomObj;        {generador aleatorio en caso que lo necesite}
maxQueueBalk:INTEGER;       {límite de Balking, si contempla Balking}
maxWaitReneg:REAL;          {límite de Reneging, si contempla Reneging}
```

```
ASK METHOD cambiarMaxQueueBalk(IN x:INTEGER);
ASK METHOD cambiarMaxWaitReneg(IN x:REAL);

ASK METHOD llegoPasajeroPiso(IN P:Piso; IN direccion:INTEGER; IN pas:Pasajero);
ASK METHOD finGoToPiso(IN A:Ascensor; IN P:Piso);
ASK METHOD finStopAtPiso(IN A:Ascensor; IN P:Piso);
ASK METHOD finSubeBajaPas(IN A:Ascensor; IN P:Piso; IN dir:INTEGER);
ASK METHOD finSubePas(IN A:Ascensor; IN P:Piso; IN dir:INTEGER);
ASK METHOD finSubeUnPas(IN A:Ascensor; IN P:Piso; IN dir:INTEGER);
ASK METHOD finBajaPas(IN A:Ascensor; IN P:Piso; IN dir:INTEGER);
ASK METHOD finGoAndStopAtPiso(IN A:Ascensor; IN P:Piso);
ASK METHOD finGoStopAndSubeBajaAtPiso(IN A:Ascensor; IN P:Piso; IN dir:INTEGER);

ASK METHOD ObjReset(); {Borra todo dato temporal restante de una corrida}
ASK METHOD ObjStart(); {Inicializa ascensores}
ASK METHOD ObjInit();  {Inicializa scheduler}
ASK METHOD ObjTerminate();
```

ASCENSOR
TELL METHOD goToPiso(IN P:Piso); TELL METHOD stopAtPiso(IN P:Piso); TELL METHOD subeBajaPas(IN dir:INTEGER); TELL METHOD subePas(IN dir:INTEGER); TELL METHOD subeUnPas(IN dir:INTEGER); TELL METHOD bajaPas(IN dir:INTEGER); TELL METHOD goAndStopAtPiso(IN P:Piso); TELL METHOD goStopAndSubeBajaAtPiso(IN P:Piso;IN dir:INTEGER);

PISO
TELL METHOD crearNuevoPasajero(); ASK METHOD disposePasajeroBalk(IN direccion:INTEGER; IN pas:Pasajero); TELL METHOD disposePasajeroReneg(IN direccion:INTEGER; IN pas:Pasajero);

Las clases de SchedulerObj brindan los recursos básicos necesarios para un manejador: su nombre, un generador de números aleatorios y los valores definidos para la corrida de Balking y Reneging, si es que se van a usar.

El interfaz es brindado por los métodos de SchedulerObj y Ascensor que mencionamos en los recuadros. Su funcionamiento es el siguiente:

llegoPasajeroPiso es invocado por *crearNuevoPasajero* de Piso para informar al manejador de la llegada de un pasajero a un piso. El manejador puede invocar en ese momento al *disposePasajeroBalk* del piso en cuestión para eliminar al pasajero si el largo de la cola supera el límite de Balking. También puede programar que se ejecute *disposePasajeroReneg* luego del tiempo límite de Reneging (solo se ejecuta si el pasajero aún no subió a un ascensor después de transcurrido ese intervalo). Finalmente, debería activarse una reevaluación del plan de ruteo (pues el estado cambió), que eventualmente llamará a alguno de los métodos de control de ascensores mencionados en el objeto Ascensor.

Estos métodos de control de ascensores le ordenan a un ascensor ir a un piso, detenerse y/o bajar y subir uno o todos los pasajeros posibles. Sus nombres son explicativos de su función precisa. Luego que el ascensor cumplió con la orden recibida con el método de control *X*, invocará al método del scheduler *finX*. De esta manera informará al manejador que la orden fue cumplida y que el estado del sistema cambió, lo que debería activar un reevaluación del plan de ruteo de ascensores.

Como los métodos *finX* sólo pueden ser invocados por sus equivalentes *X* del objeto Ascensor, que a su vez sólo pueden ser invocados por el manejador, no hace falta reescribir todos los *finX* al implementar un nuevo manejador. Sólo hace falta modificar los que se vayan a usar, pues sabemos que los otros nunca serán invocados.

El primer paso para crear el nuevo manejador es escribir su definición en el archivo Dscheduler.mod . Si hay tipos de datos o funciones auxiliares, también se los debe ingresar allí. Ej:

```
       SchedulerHUFF = OBJECT(SchedulerObj) {Scheduler HUFF Basico}
           OVERRIDE ASK METHOD finGoToPiso(IN A:Ascensor; IN P:Piso);
           OVERRIDE ASK METHOD finStopAtPiso(IN A:Ascensor; IN P:Piso);
           OVERRIDE ASK METHOD finBajaPas(IN A:Ascensor; IN P:Piso; IN dir:INTEGER);
           OVERRIDE ASK METHOD finSubeUnPas(IN A:Ascensor; IN P:Piso; IN dir:INTEGER);
           OVERRIDE ASK METHOD llegoPasajeroPiso(IN P:Piso; IN direccion:INTEGER; IN pas:Pasajero);
           OVERRIDE ASK METHOD ObjReset();
           OVERRIDE ASK METHOD ObjStart();
           OVERRIDE ASK METHOD ObjInit();  {Inicializa scheduler}
           OVERRIDE ASK METHOD ObjTerminate();
```

En este ejemplo vemos cómo definimos SchedulerHUFF como descendiente de SchedulerObj, y modificamos los métodos *finX* que vamos a utilizar. También modificamos los métodos de creación y destrucción del objeto.

Luego se debe escribir la implementación del nuevo objeto (y sus eventuales tipos y funciones auxiliares) en Ischeduler.mod , siguiendo el funcionamiento descripto en la definición del interface. Ej:

```
    ASK METHOD llegoPasajeroPiso(IN P:Piso; IN direccion:INTEGER; IN pas:Pasajero)
    VAR ped:PedidoHUFF;
        AP:AscPedidoHUFF;
        newP,pisoActual:Piso;
        pisoAct:INTEGER;
        a:ACTID;
        tiempo:REAL;
        pj:Pasajero;
    BEGIN
        pisoAct:=ASK P numero;

        IF (maxQueueBalk<ASK P largoColasPas()) {si hay balking, lo sacamos y destruimos}
            ASK P TO disposePasajeroBalk(direccion,pas);

        ELSE
            a:=TELL P TO disposePasajeroReneg(direccion,pas) IN maxWaitReneg; {seteamos el
reneging}
            ASK pas TO cambiaSchedDes(a);

            IF (NOT HayPedExtPiso(direccion, pisoAct)) AND
               (NOT HayPedExtAsigPiso(direccion, pisoAct))
               {solo actuamos si no hay pedido asignado o no de ese piso}
                NEW(ped);
                ped.piso:= pisoAct
                AddPedido(direccion, ped);

                AP:= ClosestAscLibrePark(ped.piso);
                IF AP<>NILREC {si algun ascensor no esta haciendo nada, que conteste a este pedido}

                    pisoActual:= ASK AP.ascensor pisoActual;
                    IF ASK pisoActual numero<pisoAct {va a subir}
                        IF direccion>0
                            AsignarPedidosSuben(AP,ASK pisoActual numero);
                           {va a subir luego de estar parado; le asignamos
                            al ascensor los pedidos que suben}
                            newP:=ASK edificio.pisos Next(pisoActual);
                            SetAscLibre(AP, FALSE);
                            TELL AP.ascensor TO goToPiso(newP) IN
                                     TiempoStartAtPiso(AP.ascensor, pisoActual) +
                                     TiempoGoToPiso(AP.ascensor, pisoActual, newP);
                        ELSE
                            NuevasAsignaciones();
                        END IF;
```

```
                ELSIF ASK pisoActual numero>pisoAct {va a bajar}
                    IF direccion>0
                        newP:=ASK edificio.pisos Prev(pisoActual);
                        SetAscLibre(AP, TRUE);
                        TELL AP.ascensor TO goToPiso(newP) IN
                                TiempoStartAtPiso(AP.ascensor, pisoActual) +
                                TiempoGoToPiso(AP.ascensor, pisoActual, newP);
                    ELSE
                        NuevasAsignaciones();
                    END IF;
                ELSE
                    IF direccion>0
                        ped:=RemovePedido(direccion,ASK P numero);
                        {le asignamos el pedido tope que baja}
                        AddPedidoAsigAscPiso(direccion,AP,ped);
                        AsignarPedidosSuben(AP,ASK pisoActual numero);
                        {va a subir luego de estar parado; le asignamos
                         al ascensor los pedidos que suben}
                        ASK AP.ascensor TO unpark(direccion);
                        SetAscLibre(AP, FALSE);
                        tiempo:=TiempoSubeUnPasBR(AP.ascensor, P, direccion, pas)
                        TELL AP.ascensor TO subeUnPas(direccion) IN tiempo;
                    ELSE
                        NuevasAsignaciones();
                    END IF;
                END IF;
            END IF;
        END IF;
    END IF;
END METHOD;
```

B.2) Procedimiento para Agregarlo

Una vez creado el nuevo objeto manejador, debemos integrarlo con el interfaz gráfico para que sea seleccionable por el usuario. Para esto debemos modificar dos funciones en el archivo Ischeduler.mod: *ListaSchedulers* y *Selectscheduler*.

ListaSchedulers define un array con los nombres de todos los manejadores (estos son los nombres que se muestran en el cuadro de diálogo de opciones). Para agregar un nuevo manejador se debe incrementar en uno el rango superior del array *lista* y asignar el nombre elegido para el manejador al nuevo elemento del array. Ej:

```
PROCEDURE ListaSchedulers():ListaSched
VAR lista:ListaSched;
BEGIN
    NEW (lista,1..9);
    lista[1]:="nulo";
    .
    .
    .
    lista[9]:="HUFF con B y R";
    RETURN (lista);
END PROCEDURE;
```

En este ejemplo para agregar un nuevo manejador de tipo SchedulerNUEVO y nombre *nombrenuevo* deberíamos modificar la cuarta línea para que diga “`NEW (lista,1..10);`” y agregar una línea “`lista[10]:="nombrenuevo";`” antes del RETURN.

Selectscheduler, por su parte, define la selección e inicialización apropiada del manejador a partir de los valores elegidos en el cuadro de diálogo de opciones. Para agregar un nuevo manejador debemos definir una variable del tipo del manejador y agregar una sentencia CASE para el caso del nombre del nuevo manejador definido en *ListaSchedulers*. Ej:

```
PROCEDURE SelectScheduler(IN nombre:STRING; IN semilla:INTEGER; INOUT
balk:INTEGER; INOUT reneg:REAL):BOOLEAN
VAR schnulo:SchedulerNulo;
    .
    .
    .
    schhuffbr:SchedulerHUFFBR;
    result:BOOLEAN;
BEGIN
     {segun nombre, discriminar cual tipo usar}
    CASE nombre
        WHEN "nulo":
                    NEW(schnulo);
                    IF scheduler<>NILOBJ DISPOSE(scheduler); END IF;
                    balk:=0;
                    reneg:=0.0;
                    scheduler := schnulo;
                    result:=TRUE;
        .
        .
        .
        WHEN "HUFF con B y R":
                    NEW(schhuffbr);
                    ASK (ASK schhuffbr generador) TO SetSeed(semilla);
                    ASK schhuffbr TO cambiarMaxQueueBalk(balk);
                    ASK schhuffbr TO cambiarMaxWaitReneg(reneg);
                    IF scheduler<>NILOBJ DISPOSE(scheduler); END IF;
                    scheduler := schhuffbr;
                    result:=TRUE;
        OTHERWISE
                    result:=FALSE;
    END CASE;
    RETURN result;
END PROCEDURE;
```

En este ejemplo, para agregar nuestro nuevo manejador debemos agregar una línea "`schnuevo:SchedulerNUEVO;`" a la sección VAR, y un CASE de forma:

```
WHEN "nombrenuevo":
      NEW(schnuevo);
      ASK (ASK schnuevo generador) TO SetSeed(semilla);
                                    {si se requieren números aleatorios}

      [ASK schnuevo TO cambiarMaxQueueBalk(balk); {si se usa Balking}
            O
      balk:=0; {si no se usa Balking}]

      [ASK schnuevo TO cambiarMaxWaitReneg(reneg); {si se usa Reneging}
            O
      reneg:=0.0; {si no se usa Reneging}]
```

```
IF scheduler<>NILOBJ DISPOSE(scheduler); END IF;
scheduler := schnuevo; {se asigna nuestra nueva instancia
                        inicializada de SchedulerNUEVO a la
                        variable scheduler}
result:=TRUE; {devolvemos el éxito de la operación}
```

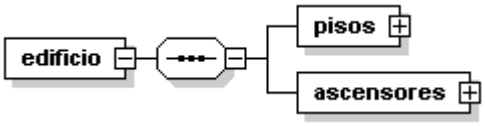
APENDICE C: Especificación del Esquema de la Entrada

Schema **EdificioSim.xsd**

schema location: **EdificioSim.xsd**
targetNamespace: **MODSIM_simulador_edificio_1.00**

Elements	Complex types	Simple types
edificio	**tipo_ascensor**	**tipo_decimal**
	tipo_distribucion	
	tipo_piso	

element **edificio**

diagram	
namespace	MODSIM_simulador_edificio_1.00
children	**pisos** **ascensores**

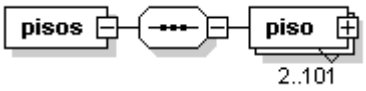
attributes	Name	Type	Use	Default	Fixed	Annotation
	nombre	xs:string	optional			
	segundos_periodo	xs:integer	required			

source

```
<xs:element name="edificio">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="pisos">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="piso" type="tipo_piso" minOccurs="2" maxOccurs="101">
       <xs:unique name="un_inicio_generador">
        <xs:selector xpath="generadores/intervalo"/>
        <xs:field xpath="@segundo_inicio"/>
       </xs:unique>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
    <xs:key name="key_numero_piso">
     <xs:selector xpath="piso"/>
     <xs:field xpath="numero"/>
    </xs:key>
   </xs:element>
   <xs:element name="ascensores">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="ascensor" type="tipo_ascensor" maxOccurs="10">
       <xs:unique name="un_inicio_parada_posible">
        <xs:selector xpath="paradas_posibles/intervalo"/>
        <xs:field xpath="@segundo_inicio"/>
       </xs:unique>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
    <xs:key name="key_numero_ascensor">
     <xs:selector xpath="ascensor"/>
     <xs:field xpath="numero"/>
```

```
      </xs:key>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="nombre" type="xs:string" use="optional"/>
  <xs:attribute name="segundos_periodo" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:minInclusive value="60"/>
        <xs:maxInclusive value="604800"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
</xs:element>
```

element **edificio/pisos**

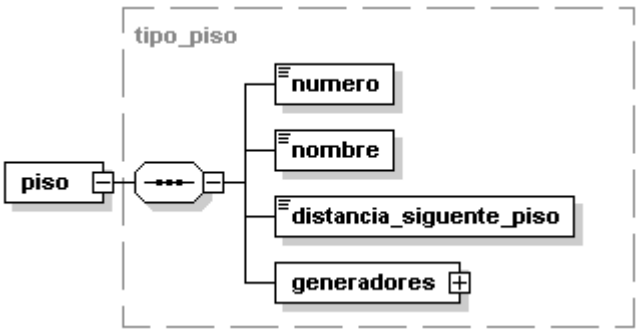
diagram	
namespace	MODSIM_simulador_edificio_1.00
children	**piso**

identity constraints	Name	Refer	Selector	Field(s)
key	key_numero_piso		piso	numero

source

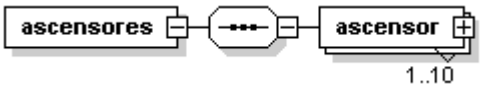
```
<xs:element name="pisos">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="piso" type="tipo_piso" minOccurs="2" maxOccurs="101">
    <xs:unique name="un_inicio_generador">
     <xs:selector xpath="generadores/intervalo"/>
     <xs:field xpath="@segundo_inicio"/>
    </xs:unique>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:key name="key_numero_piso">
  <xs:selector xpath="piso"/>
  <xs:field xpath="numero"/>
 </xs:key>
</xs:element>
```

element **edificio/pisos/piso**

diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**tipo_piso**

children	**numero nombre distancia_siguiente_piso generadores**				
identity constraints		Name	Refer	Selector	Field(s)
	unique	un_inicio_generador		generadores/intervalo	@segundo_inicio
source	<xs:element name="piso" type="tipo_piso" minOccurs="2" maxOccurs="101"> <xs:unique name="un_inicio_generador"> <xs:selector xpath="generadores/intervalo"/> <xs:field xpath="@segundo_inicio"/> </xs:unique> </xs:element>				

element **edificio/ascensores**

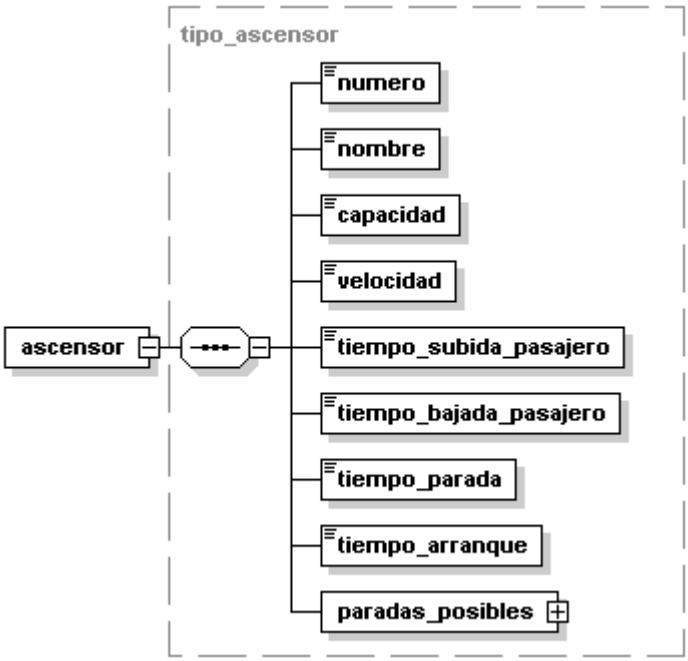
diagram	
namespace	MODSIM_simulador_edificio_1.00
children	**ascensor**

identity constraints	Name	Refer	Selector	Field(s)
key	key_numero_ascensor		ascensor	numero

source

```
<xs:element name="ascensores">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="ascensor" type="tipo_ascensor" maxOccurs="10">
    <xs:unique name="un_inicio_parada_posible">
     <xs:selector xpath="paradas_posibles/intervalo"/>
     <xs:field xpath="@segundo_inicio"/>
    </xs:unique>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:key name="key_numero_ascensor">
  <xs:selector xpath="ascensor"/>
  <xs:field xpath="numero"/>
 </xs:key>
</xs:element>
```

element **edificio/ascensores/ascensor**

diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**tipo_ascensor**
children	**numero nombre capacidad velocidad tiempo_subida_pasajero tiempo_bajada_pasajero tiempo_parada tiempo_arranque paradas_posibles**

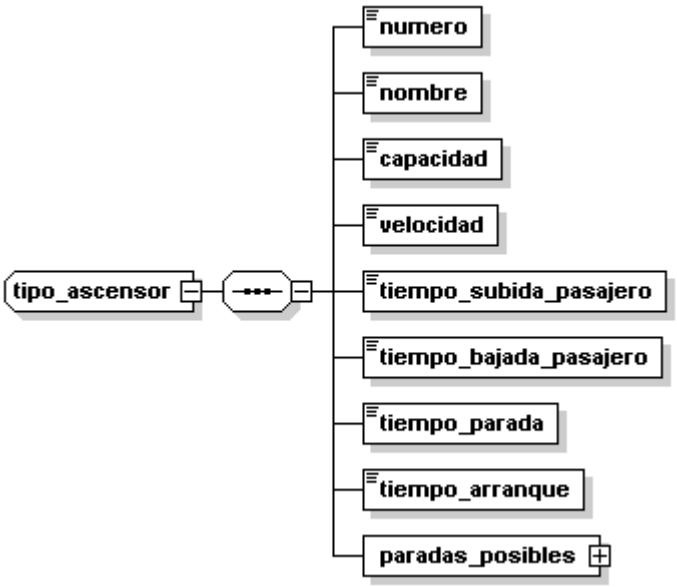
identity constraints		Name	Refer	Selector	Field(s)
	unique	un_inicio_parada_posible		paradas_posibles/intervalo	@segundo_inicio

source

```
<xs:element name="ascensor" type="tipo_ascensor" maxOccurs="10">
  <xs:unique name="un_inicio_parada_posible">
    <xs:selector xpath="paradas_posibles/intervalo"/>
    <xs:field xpath="@segundo_inicio"/>
  </xs:unique>
</xs:element>
```

complexType **tipo_ascensor**

diagram



namespace: MODSIM_simulador_edificio_1.00

children: **numero** **nombre** **capacidad** **velocidad** **tiempo_subida_pasajero** **tiempo_bajada_pasajero** **tiempo_parada** **tiempo_arranque** **paradas_posibles**

used by: element **edificio/ascensores/ascensor**

source

```
<xs:complexType name="tipo_ascensor">
 <xs:sequence>
  <xs:element name="numero" type="xs:integer"/>
  <xs:element name="nombre" type="xs:string" default="undef"/>
  <xs:element name="capacidad" type="xs:integer"/>
  <xs:element name="velocidad" type="tipo_decimal"/>
  <xs:element name="tiempo_subida_pasajero" type="tipo_decimal"/>
  <xs:element name="tiempo_bajada_pasajero" type="tipo_decimal"/>
  <xs:element name="tiempo_parada" type="tipo_decimal"/>
  <xs:element name="tiempo_arranque" type="tipo_decimal"/>
  <xs:element name="paradas_posibles">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="intervalo" maxOccurs="unbounded">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="parada" type="xs:integer" minOccurs="0" maxOccurs="100">
         <xs:keyref name="ref_numero_parada" refer="key_numero_piso">
          <xs:selector xpath="."/>
          <xs:field xpath="."/>
         </xs:keyref>
        </xs:element>
       </xs:sequence>
       <xs:attribute name="segundo_inicio" use="required">
        <xs:simpleType>
         <xs:restriction base="xs:integer">
          <xs:minInclusive value="0"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:attribute>
      </xs:complexType>
      <xs:unique name="un_numero_parada">
       <xs:selector xpath="parada"/>
       <xs:field xpath="."/>
```

	</xs:unique> </xs:element> </xs:sequence> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType>

element **tipo_ascensor/numero**

diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**xs:integer**
source	<xs:element name="numero" type="xs:integer"/>

element **tipo_ascensor/nombre**

diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**xs:string**
source	<xs:element name="nombre" type="xs:string" default="undef"/>

element **tipo_ascensor/capacidad**

diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**xs:integer**
source	<xs:element name="capacidad" type="xs:integer"/>

element **tipo_ascensor/velocidad**

diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**tipo_decimal**
facets	fractionDigits 3
source	<xs:element name="velocidad" type="tipo_decimal"/>

element **tipo_ascensor/tiempo_subida_pasajero**

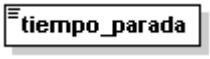
diagram	
namespace	MODSIM_simulador_edificio_1.00

type	**tipo_decimal**
facets	fractionDigits 3
source	<xs:element name="tiempo_subida_pasajero" type="tipo_decimal"/>

element **tipo_ascensor/tiempo_bajada_pasajero**

diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**tipo_decimal**
facets	fractionDigits 3
source	<xs:element name="tiempo_bajada_pasajero" type="tipo_decimal"/>

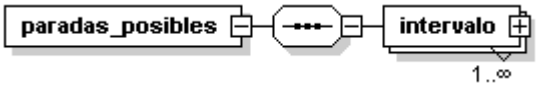
element **tipo_ascensor/tiempo_parada**

diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**tipo_decimal**
facets	fractionDigits 3
source	<xs:element name="tiempo_parada" type="tipo_decimal"/>

element **tipo_ascensor/tiempo_arranque**

diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**tipo_decimal**
facets	fractionDigits 3
source	<xs:element name="tiempo_arranque" type="tipo_decimal"/>

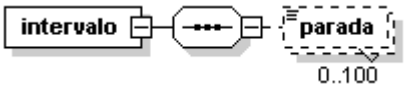
element **tipo_ascensor/paradas_posibles**

diagram	
namespace	MODSIM_simulador_edificio_1.00
children	**intervalo**
source	<xs:element name="paradas_posibles"> <xs:complexType> <xs:sequence> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="parada" type="xs:integer" minOccurs="0" maxOccurs="100"> <xs:keyref name="ref_numero_parada" refer="key_numero_piso">

```
            <xs:selector xpath="."/>
            <xs:field xpath="."/>
          </xs:keyref>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="segundo_inicio" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
    <xs:unique name="un_numero_parada">
      <xs:selector xpath="parada"/>
      <xs:field xpath="."/>
    </xs:unique>
  </xs:element>
 </xs:sequence>
</xs:complexType>
</xs:element>
```

element **tipo_ascensor/paradas_posibles/intervalo**

diagram



namespace: MODSIM_simulador_edificio_1.00

children: **parada**

attributes

Name	Type	Use	Default	Fixed	Annotation
segundo_inicio	xs:integer	required			

identity constraints

	Name	Refer	Selector	Field(s)
unique	un_numero_parada		parada	.

source

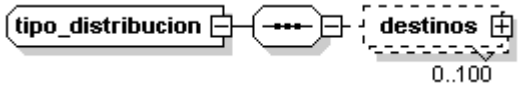
```
<xs:element name="intervalo" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="parada" type="xs:integer" minOccurs="0" maxOccurs="100">
    <xs:keyref name="ref_numero_parada" refer="key_numero_piso">
     <xs:selector xpath="."/>
     <xs:field xpath="."/>
    </xs:keyref>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="segundo_inicio" use="required">
   <xs:simpleType>
    <xs:restriction base="xs:integer">
     <xs:minInclusive value="0"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
 <xs:unique name="un_numero_parada">
  <xs:selector xpath="parada"/>
  <xs:field xpath="."/>
 </xs:unique>
</xs:element>
```

element **tipo_ascensor/paradas_posibles/intervalo/parada**

diagram

namespace	MODSIM_simulador_edificio_1.00
type	**xs:integer**
identity constraints	keyref — Name: ref_numero_parada — Refer: key_numero_piso — Selector: . — Field(s): .
source	<xs:element name="parada" type="xs:integer" minOccurs="0" maxOccurs="100"> <xs:keyref name="ref_numero_parada" refer="key_numero_piso"> <xs:selector xpath="."/> <xs:field xpath="."/> </xs:keyref> </xs:element>

complexType **tipo_distribucion**

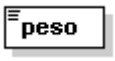
diagram	
namespace	MODSIM_simulador_edificio_1.00
children	**destinos**
used by	element **tipo_piso/generadores/intervalo/distribucion**
source	<xs:complexType name="tipo_distribucion"> <xs:sequence> <xs:element name="destinos" minOccurs="0" maxOccurs="100"> <xs:complexType> <xs:sequence> <xs:element name="peso" type="tipo_decimal"/> </xs:sequence> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:complexType> <xs:keyref name="ref_numero_destino" refer="key_numero_piso"> <xs:selector xpath="."/> <xs:field xpath="@numero"/> </xs:keyref> </xs:element> </xs:sequence> </xs:complexType>

element **tipo_distribucion/destinos**

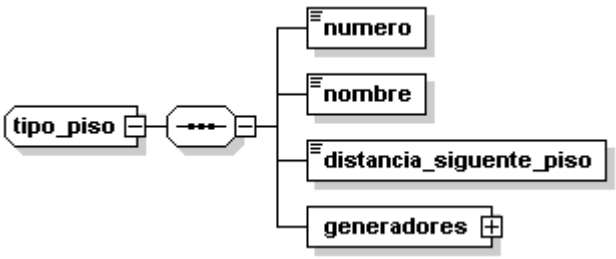
diagram	
namespace	MODSIM_simulador_edificio_1.00
children	**peso**
attributes	Name: numero — Type: xs:integer — Use: required — Default — Fixed — Annotation
identity constraints	keyref — Name: ref_numero_destino — Refer: key_numero_piso — Selector: . — Field(s): @numero
source	<xs:element name="destinos" minOccurs="0" maxOccurs="100"> <xs:complexType> <xs:sequence> <xs:element name="peso" type="tipo_decimal"/> </xs:sequence> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:complexType> <xs:keyref name="ref_numero_destino" refer="key_numero_piso"> <xs:selector xpath="."/>

```
    <xs:field xpath="@numero"/>
  </xs:keyref>
</xs:element>
```

element **tipo_distribucion/destinos/peso**

diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**tipo_decimal**
facets	fractionDigits 3
source	`<xs:element name="peso" type="tipo_decimal"/>`

complexType **tipo_piso**

diagram	
namespace	MODSIM_simulador_edificio_1.00
children	**numero** **nombre** **distancia_siguente_piso** **generadores**
used by	element **edificio/pisos/piso**
source	

```
<xs:complexType name="tipo_piso">
  <xs:sequence>
    <xs:element name="numero" type="xs:integer"/>
    <xs:element name="nombre" type="xs:string" default="undef"/>
    <xs:element name="distancia_siguente_piso" type="tipo_decimal"/>
    <xs:element name="generadores">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="intervalo" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="frecuencia" type="tipo_decimal"/>
                <xs:element name="distribucion" type="tipo_distribucion">
                  <xs:unique name="un_numero_destino">
                    <xs:selector xpath="destinos"/>
                    <xs:field xpath="@numero"/>
                  </xs:unique>
                </xs:element>
              </xs:sequence>
              <xs:attribute name="segundo_inicio" use="required">
                <xs:simpleType>
                  <xs:restriction base="xs:integer">
                    <xs:minInclusive value="0"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
```

	</xs:sequence> </xs:complexType>

element **tipo_piso/numero**

diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**xs:integer**
source	<xs:element name="numero" type="xs:integer"/>

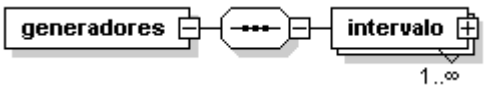
element **tipo_piso/nombre**

diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**xs:string**
source	<xs:element name="nombre" type="xs:string" default="undef"/>

element **tipo_piso/distancia_siguente_piso**

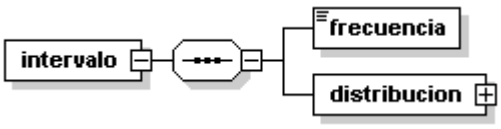
diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**tipo_decimal**
facets	fractionDigits 3
source	<xs:element name="distancia_siguente_piso" type="tipo_decimal"/>

element **tipo_piso/generadores**

diagram	
namespace	MODSIM_simulador_edificio_1.00
children	**intervalo**
source	<xs:element name="generadores"> <xs:complexType> <xs:sequence> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="frecuencia" type="tipo_decimal"/> <xs:element name="distribucion" type="tipo_distribucion"> <xs:unique name="un_numero_destino"> <xs:selector xpath="destinos"/> <xs:field xpath="@numero"/> </xs:unique> </xs:element> </xs:sequence> <xs:attribute name="segundo_inicio" use="required">

```
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:minInclusive value="0"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

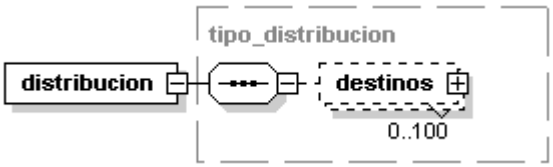
element **tipo_piso/generadores/intervalo**

diagram	
namespace	MODSIM_simulador_edificio_1.00
children	**frecuencia** **distribucion**
attributes	Name: segundo_inicio; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:
source	<xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="frecuencia" type="tipo_decimal"/> <xs:element name="distribucion" type="tipo_distribucion"> <xs:unique name="un_numero_destino"> <xs:selector xpath="destinos"/> <xs:field xpath="@numero"/> </xs:unique> </xs:element> </xs:sequence> <xs:attribute name="segundo_inicio" use="required"> <xs:simpleType> <xs:restriction base="xs:integer"> <xs:minInclusive value="0"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element>

element **tipo_piso/generadores/intervalo/frecuencia**

diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**tipo_decimal**
facets	fractionDigits 3
source	<xs:element name="frecuencia" type="tipo_decimal"/>

element **tipo_piso/generadores/intervalo/distribucion**

diagram	
namespace	MODSIM_simulador_edificio_1.00
type	**tipo_distribucion**
children	**destinos**
identity constraints	unique — Name: un_numero_destino — Refer: — Selector: destinos — Field(s): @numero
source	<xs:element name="distribucion" type="tipo_distribucion"> <xs:unique name="un_numero_destino"> <xs:selector xpath="destinos"/> <xs:field xpath="@numero"/> </xs:unique> </xs:element>

simpleType **tipo_decimal**

namespace	MODSIM_simulador_edificio_1.00
type	restriction of **xs:decimal**
used by	elements **tipo_piso/distancia_siguente_piso** **tipo_piso/generadores/intervalo/frecuencia** **tipo_distribucion/destinos/peso** **tipo_ascensor/tiempo_arranque** **tipo_ascensor/tiempo_bajada_pasajero** **tipo_ascensor/tiempo_parada** **tipo_ascensor/tiempo_subida_pasajero** **tipo_ascensor/velocidad**
facets	fractionDigits 3
source	<xs:simpleType name="tipo_decimal"> <xs:restriction base="xs:decimal"> <xs:fractionDigits value="3"/> </xs:restriction> </xs:simpleType>

APENDICE D: Especificación del Esquema de la Salida

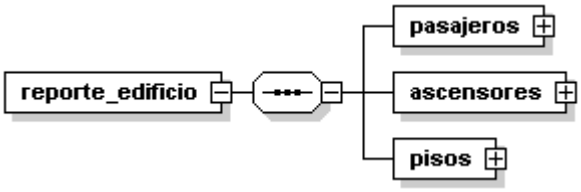
Schema **EdificioRep.xsd**

schema location: **EdificioRep.xsd**
targetNamespace: **MODSIM_reporte_simulador_edificio_1.00**

Elements
reporte_edificio

element **reporte_edificio**

diagram



namespace: MODSIM_reporte_simulador_edificio_1.00

children: **pasajeros** **ascensores** **pisos**

attributes

Name	Type	Use	Default	Fixed	Annotation
nombre	xs:string	required			
cantidad_intervalos	xs:integer	required			
semilla_randomica	xs:integer	required			
duracion_total	xs:float	required			
segundos_periodo	xs:integer	required			
nombre_scheduler	xs:string	required			
semilla_scheduler	xs:integer	required			
balking_scheduler	xs:integer	optional			
reneging_scheduler	xs:float	optional			

source

```
<xs:element name="reporte_edificio">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="pasajeros">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="espera_media">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="global" type="xs:float"/>
         <xs:element name="intervalo" maxOccurs="unbounded">
          <xs:complexType>
           <xs:simpleContent>
            <xs:extension base="xs:float">
             <xs:attribute name="numero" type="xs:integer" use="required"/>
            </xs:extension>
           </xs:simpleContent>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
```

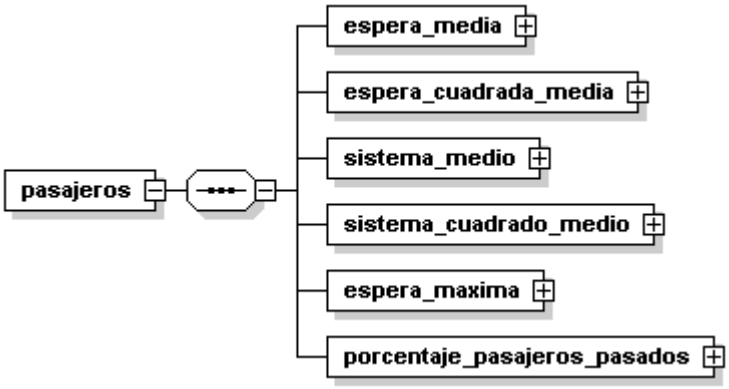
```
    </xs:complexType>
  </xs:element>
  <xs:element name="espera_cuadrada_media">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="global" type="xs:float"/>
        <xs:element name="intervalo" maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:float">
                <xs:attribute name="numero" type="xs:integer" use="required"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="sistema_medio">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="global" type="xs:float"/>
        <xs:element name="intervalo" maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:float">
                <xs:attribute name="numero" type="xs:integer" use="required"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="sistema_cuadrado_medio">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="global" type="xs:float"/>
        <xs:element name="intervalo" maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:float">
                <xs:attribute name="numero" type="xs:integer" use="required"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="espera_maxima">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="global" type="xs:float"/>
        <xs:element name="intervalo" maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:float">
                <xs:attribute name="numero" type="xs:integer" use="required"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="porcentaje_pasajeros_pasados">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="global" type="xs:float"/>
```

```
            <xs:element name="intervalo" maxOccurs="unbounded">
              <xs:complexType>
                <xs:simpleContent>
                  <xs:extension base="xs:float">
                    <xs:attribute name="numero" type="xs:integer" use="required"/>
                  </xs:extension>
                </xs:simpleContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="tiempo_limite" type="xs:float" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ascensores">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ocupacion_media">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="ascensor" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="global" type="xs:float"/>
                  <xs:element name="intervalo" maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:simpleContent>
                        <xs:extension base="xs:float">
                          <xs:attribute name="numero" type="xs:integer" use="required"/>
                        </xs:extension>
                      </xs:simpleContent>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
                <xs:attribute name="numero" type="xs:integer" use="required"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="cantidad_paradas">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="ascensor" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="global" type="xs:float"/>
                  <xs:element name="intervalo" maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:simpleContent>
                        <xs:extension base="xs:float">
                          <xs:attribute name="numero" type="xs:integer" use="required"/>
                        </xs:extension>
                      </xs:simpleContent>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
                <xs:attribute name="numero" type="xs:integer" use="required"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="pisos">
  <xs:complexType>
```

```
        <xs:sequence>
          <xs:element name="largo_cola_medio">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="piso" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="global" type="xs:float"/>
                      <xs:element name="intervalo" maxOccurs="unbounded">
                        <xs:complexType>
                          <xs:simpleContent>
                            <xs:extension base="xs:float">
                              <xs:attribute name="numero" type="xs:integer" use="required"/>
                            </xs:extension>
                          </xs:simpleContent>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                    <xs:attribute name="numero" type="xs:integer" use="required"/>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="largo_cola_maximo">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="piso" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="global" type="xs:float"/>
                      <xs:element name="intervalo" maxOccurs="unbounded">
                        <xs:complexType>
                          <xs:simpleContent>
                            <xs:extension base="xs:float">
                              <xs:attribute name="numero" type="xs:integer" use="required"/>
                            </xs:extension>
                          </xs:simpleContent>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                    <xs:attribute name="numero" type="xs:integer" use="required"/>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="cantidad_pasajeros_generados">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="piso" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="global" type="xs:float"/>
                      <xs:element name="intervalo" maxOccurs="unbounded">
                        <xs:complexType>
                          <xs:simpleContent>
                            <xs:extension base="xs:float">
                              <xs:attribute name="numero" type="xs:integer" use="required"/>
                            </xs:extension>
                          </xs:simpleContent>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                    <xs:attribute name="numero" type="xs:integer" use="required"/>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
```

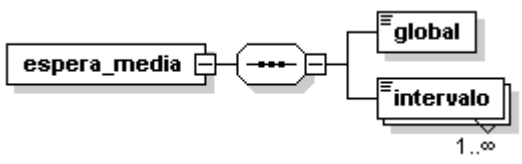
```
        <xs:element name="cantidad_pasajeros_balked" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="piso" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="global" type="xs:integer"/>
                    <xs:element name="intervalo" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:simpleContent>
                          <xs:extension base="xs:integer">
                            <xs:attribute name="numero" type="xs:integer" use="required"/>
                          </xs:extension>
                        </xs:simpleContent>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="numero" type="xs:integer" use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="cantidad_pasajeros_reneged" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="piso" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="global" type="xs:integer"/>
                    <xs:element name="intervalo" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:simpleContent>
                          <xs:extension base="xs:integer">
                            <xs:attribute name="numero" type="xs:integer" use="required"/>
                          </xs:extension>
                        </xs:simpleContent>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="numero" type="xs:integer" use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:attribute name="nombre" type="xs:string" use="required"/>
  <xs:attribute name="cantidad_intervalos" type="xs:integer" use="required"/>
  <xs:attribute name="semilla_randomica" type="xs:integer" use="required"/>
  <xs:attribute name="duracion_total" type="xs:float" use="required"/>
  <xs:attribute name="segundos_periodo" type="xs:integer" use="required"/>
  <xs:attribute name="nombre_scheduler" type="xs:string" use="required"/>
  <xs:attribute name="semilla_scheduler" type="xs:integer" use="required"/>
  <xs:attribute name="balking_scheduler" type="xs:integer" use="optional"/>
  <xs:attribute name="reneging_scheduler" type="xs:float" use="optional"/>
 </xs:complexType>
</xs:element>
```

element **reporte_edificio/pasajeros**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**espera_media espera_cuadrada_media sistema_medio sistema_cuadrado_medio espera_maxima porcentaje_pasajeros_pasados**
source	<xs:element name="pasajeros"> <xs:complexType> <xs:sequence> <xs:element name="espera_media"> <xs:complexType> <xs:sequence> <xs:element name="global" type="xs:float"/> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:float"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </xs:element> <xs:element name="espera_cuadrada_media"> <xs:complexType> <xs:sequence> <xs:element name="global" type="xs:float"/> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:float"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </xs:element> <xs:element name="sistema_medio"> <xs:complexType> <xs:sequence> <xs:element name="global" type="xs:float"/> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:float"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>

```
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="sistema_cuadrado_medio">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="global" type="xs:float"/>
          <xs:element name="intervalo" maxOccurs="unbounded">
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base="xs:float">
                  <xs:attribute name="numero" type="xs:integer" use="required"/>
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="espera_maxima">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="global" type="xs:float"/>
          <xs:element name="intervalo" maxOccurs="unbounded">
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base="xs:float">
                  <xs:attribute name="numero" type="xs:integer" use="required"/>
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="porcentaje_pasajeros_pasados">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="global" type="xs:float"/>
          <xs:element name="intervalo" maxOccurs="unbounded">
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base="xs:float">
                  <xs:attribute name="numero" type="xs:integer" use="required"/>
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="tiempo_limite" type="xs:float" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **reporte_edificio/pasajeros/espera_media**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**global** **intervalo**

source	<xs:element name="espera_media"> <xs:complexType> <xs:sequence> <xs:element name="global" type="xs:float"/> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:float"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </xs:element>

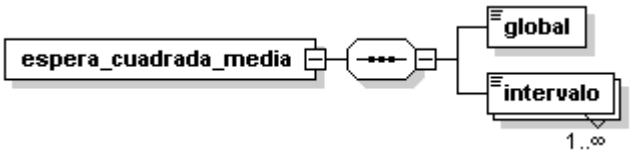
element **reporte_edificio/pasajeros/espera_media/global**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	**xs:float**
source	<xs:element name="global" type="xs:float"/>

element **reporte_edificio/pasajeros/espera_media/intervalo**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	extension of **xs:float**
attributes	Name: numero; Type: xs:integer; Use: required; Default; Fixed; Annotation
source	<xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:float"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>

element **reporte_edificio/pasajeros/espera_cuadrada_media**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**global** **intervalo**
source	<xs:element name="espera_cuadrada_media"> <xs:complexType>

```
    <xs:sequence>
      <xs:element name="global" type="xs:float"/>
      <xs:element name="intervalo" maxOccurs="unbounded">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:float">
              <xs:attribute name="numero" type="xs:integer" use="required"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

element **reporte_edificio/pasajeros/espera_cuadrada_media/global**

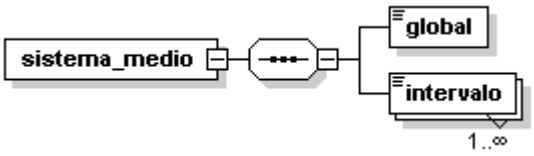
diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	**xs:float**
source	`<xs:element name="global" type="xs:float"/>`

element **reporte_edificio/pasajeros/espera_cuadrada_media/intervalo**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	extension of **xs:float**
attributes	Name: numero; Type: xs:integer; Use: required; Default; Fixed; Annotation

source	`<xs:element name="intervalo" maxOccurs="unbounded">`
	`  <xs:complexType>`
	`  <xs:simpleContent>`
	`  <xs:extension base="xs:float">`
	`  <xs:attribute name="numero" type="xs:integer" use="required"/>`
	`  </xs:extension>`
	`  </xs:simpleContent>`
	`  </xs:complexType>`
	`</xs:element>`

element **reporte_edificio/pasajeros/sistema_medio**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**global intervalo**

source	`<xs:element name="sistema_medio">`
	`  <xs:complexType>`
	`  <xs:sequence>`
	`  <xs:element name="global" type="xs:float"/>`
	`  <xs:element name="intervalo" maxOccurs="unbounded">`

```
      <xs:complexType>
       <xs:simpleContent>
        <xs:extension base="xs:float">
         <xs:attribute name="numero" type="xs:integer" use="required"/>
        </xs:extension>
       </xs:simpleContent>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
</xs:element>
```

element **reporte_edificio/pasajeros/sistema_medio/global**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	**xs:float**
source	`<xs:element name="global" type="xs:float"/>`

element **reporte_edificio/pasajeros/sistema_medio/intervalo**

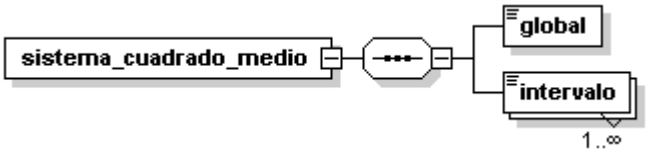
diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	extension of **xs:float**

attributes	Name	Type	Use	Default	Fixed	Annotation
	numero	xs:integer	required			

source

```
<xs:element name="intervalo" maxOccurs="unbounded">
 <xs:complexType>
  <xs:simpleContent>
   <xs:extension base="xs:float">
    <xs:attribute name="numero" type="xs:integer" use="required"/>
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

element **reporte_edificio/pasajeros/sistema_cuadrado_medio**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**global** **intervalo**

source

```
<xs:element name="sistema_cuadrado_medio">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="global" type="xs:float"/>
   <xs:element name="intervalo" maxOccurs="unbounded">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:float">
```

```
            <xs:attribute name="numero" type="xs:integer" use="required"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **reporte_edificio/pasajeros/sistema_cuadrado_medio/global**

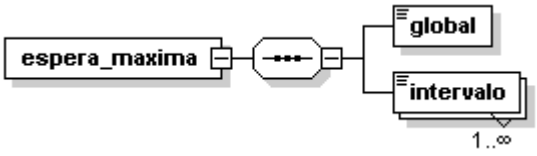
diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	**xs:float**
source	`<xs:element name="global" type="xs:float"/>`

element **reporte_edificio/pasajeros/sistema_cuadrado_medio/intervalo**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	extension of **xs:float**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:

source	`<xs:element name="intervalo" maxOccurs="unbounded">`
	`<xs:complexType>`
	`<xs:simpleContent>`
	`<xs:extension base="xs:float">`
	`<xs:attribute name="numero" type="xs:integer" use="required"/>`
	`</xs:extension>`
	`</xs:simpleContent>`
	`</xs:complexType>`
	`</xs:element>`

element **reporte_edificio/pasajeros/espera_maxima**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**global** **intervalo**

source	`<xs:element name="espera_maxima">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="global" type="xs:float"/>`
	`<xs:element name="intervalo" maxOccurs="unbounded">`
	`<xs:complexType>`
	`<xs:simpleContent>`
	`<xs:extension base="xs:float">`
	`<xs:attribute name="numero" type="xs:integer" use="required"/>`
	`</xs:extension>`
	`</xs:simpleContent>`

```
      </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

element **reporte_edificio/pasajeros/espera_maxima/global**

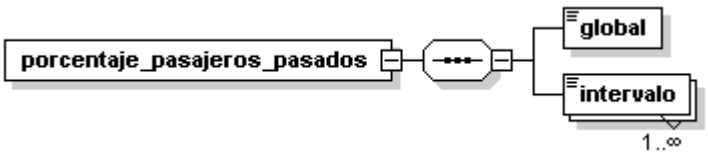
diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	**xs:float**
source	`<xs:element name="global" type="xs:float"/>`

element **reporte_edificio/pasajeros/espera_maxima/intervalo**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	extension of **xs:float**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:

source	`<xs:element name="intervalo" maxOccurs="unbounded">`
	`<xs:complexType>`
	`<xs:simpleContent>`
	`<xs:extension base="xs:float">`
	`<xs:attribute name="numero" type="xs:integer" use="required"/>`
	`</xs:extension>`
	`</xs:simpleContent>`
	`</xs:complexType>`
	`</xs:element>`

element **reporte_edificio/pasajeros/porcentaje_pasajeros_pasados**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**global** **intervalo**
attributes	Name: tiempo_limite; Type: xs:float; Use: required; Default: ; Fixed: ; Annotation:

source	`<xs:element name="porcentaje_pasajeros_pasados">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="global" type="xs:float"/>`
	`<xs:element name="intervalo" maxOccurs="unbounded">`
	`<xs:complexType>`
	`<xs:simpleContent>`
	`<xs:extension base="xs:float">`
	`<xs:attribute name="numero" type="xs:integer" use="required"/>`
	`</xs:extension>`
	`</xs:simpleContent>`

```
      </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="tiempo_limite" type="xs:float" use="required"/>
  </xs:complexType>
</xs:element>
```

element **reporte_edificio/pasajeros/porcentaje_pasajeros_pasados/global**

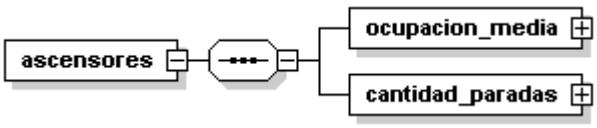
diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	**xs:float**
source	`<xs:element name="global" type="xs:float"/>`

element **reporte_edificio/pasajeros/porcentaje_pasajeros_pasados/intervalo**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	extension of **xs:float**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:

source	`<xs:element name="intervalo" maxOccurs="unbounded">`
	`<xs:complexType>`
	`<xs:simpleContent>`
	`<xs:extension base="xs:float">`
	`<xs:attribute name="numero" type="xs:integer" use="required"/>`
	`</xs:extension>`
	`</xs:simpleContent>`
	`</xs:complexType>`
	`</xs:element>`

element **reporte_edificio/ascensores**

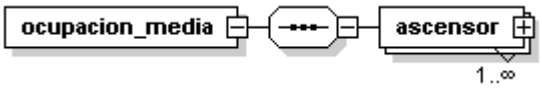
diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**ocupacion_media cantidad_paradas**

source	`<xs:element name="ascensores">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="ocupacion_media">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="ascensor" maxOccurs="unbounded">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="global" type="xs:float"/>`
	`<xs:element name="intervalo" maxOccurs="unbounded">`
	`<xs:complexType>`
	`<xs:simpleContent>`
	`<xs:extension base="xs:float">`

```
                    <xs:attribute name="numero" type="xs:integer" use="required"/>
                  </xs:extension>
                </xs:simpleContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="numero" type="xs:integer" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="cantidad_paradas">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="ascensor" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="global" type="xs:float"/>
       <xs:element name="intervalo" maxOccurs="unbounded">
        <xs:complexType>
         <xs:simpleContent>
          <xs:extension base="xs:float">
           <xs:attribute name="numero" type="xs:integer" use="required"/>
          </xs:extension>
         </xs:simpleContent>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
      <xs:attribute name="numero" type="xs:integer" use="required"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:element>
```

element **reporte_edificio/ascensores/ocupacion_media**

diagram



namespace: MODSIM_reporte_simulador_edificio_1.00

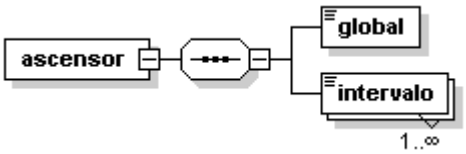
children: **ascensor**

source

```
<xs:element name="ocupacion_media">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="ascensor" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="global" type="xs:float"/>
      <xs:element name="intervalo" maxOccurs="unbounded">
       <xs:complexType>
        <xs:simpleContent>
         <xs:extension base="xs:float">
          <xs:attribute name="numero" type="xs:integer" use="required"/>
         </xs:extension>
        </xs:simpleContent>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
     <xs:attribute name="numero" type="xs:integer" use="required"/>
    </xs:complexType>
```

```
    </xs:element>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

element **reporte_edificio/ascensores/ocupacion_media/ascensor**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**global intervalo**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:
source	<xs:element name="ascensor" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="global" type="xs:float"/> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:float"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element> </xs:sequence> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:complexType> </xs:element>

element **reporte_edificio/ascensores/ocupacion_media/ascensor/global**

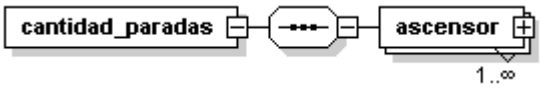
diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	**xs:float**
source	<xs:element name="global" type="xs:float"/>

element **reporte_edificio/ascensores/ocupacion_media/ascensor/intervalo**

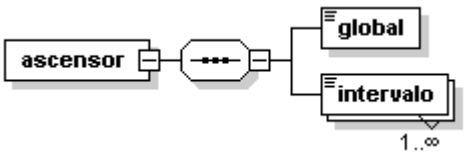
diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	extension of **xs:float**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:
source	<xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:float">

```
      <xs:attribute name="numero" type="xs:integer" use="required"/>
    </xs:extension>
   </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

element **reporte_edificio/ascensores/cantidad_paradas**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**ascensor**
source	<xs:element name="cantidad_paradas"> <xs:complexType> <xs:sequence> <xs:element name="ascensor" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="global" type="xs:float"/> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:float"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element> </xs:sequence> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </xs:element>

element **reporte_edificio/ascensores/cantidad_paradas/ascensor**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**global** **intervalo**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:
source	<xs:element name="ascensor" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="global" type="xs:float"/> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:float"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent>

```
      </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="numero" type="xs:integer" use="required"/>
  </xs:complexType>
</xs:element>
```

element **reporte_edificio/ascensores/cantidad_paradas/ascensor/global**

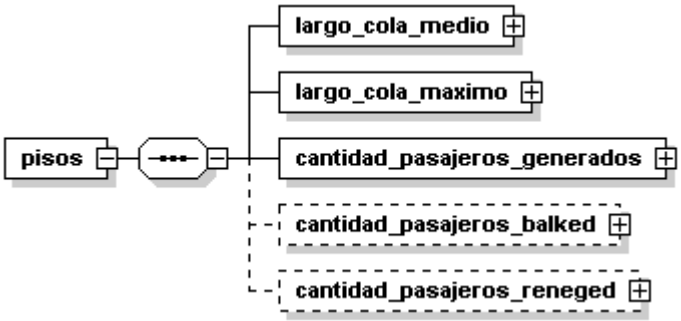
diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	**xs:float**
source	`<xs:element name="global" type="xs:float"/>`

element **reporte_edificio/ascensores/cantidad_paradas/ascensor/intervalo**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	extension of **xs:float**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:

source	`<xs:element name="intervalo" maxOccurs="unbounded">`
	`<xs:complexType>`
	`<xs:simpleContent>`
	`<xs:extension base="xs:float">`
	`<xs:attribute name="numero" type="xs:integer" use="required"/>`
	`</xs:extension>`
	`</xs:simpleContent>`
	`</xs:complexType>`
	`</xs:element>`

element **reporte_edificio/pisos**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	largo_cola_medio largo_cola_maximo cantidad_pasajeros_generados cantidad_pasajeros_balked cantidad_pasajeros_reneged

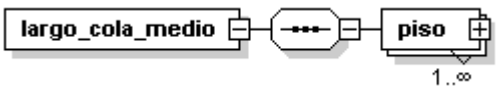
source	`<xs:element name="pisos">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="largo_cola_medio">`
	`<xs:complexType>`

```
      <xs:sequence>
       <xs:element name="piso" maxOccurs="unbounded">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="global" type="xs:float"/>
          <xs:element name="intervalo" maxOccurs="unbounded">
           <xs:complexType>
            <xs:simpleContent>
             <xs:extension base="xs:float">
              <xs:attribute name="numero" type="xs:integer" use="required"/>
             </xs:extension>
            </xs:simpleContent>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
         <xs:attribute name="numero" type="xs:integer" use="required"/>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="largo_cola_maximo">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="piso" maxOccurs="unbounded">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="global" type="xs:float"/>
          <xs:element name="intervalo" maxOccurs="unbounded">
           <xs:complexType>
            <xs:simpleContent>
             <xs:extension base="xs:float">
              <xs:attribute name="numero" type="xs:integer" use="required"/>
             </xs:extension>
            </xs:simpleContent>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
         <xs:attribute name="numero" type="xs:integer" use="required"/>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="cantidad_pasajeros_generados">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="piso" maxOccurs="unbounded">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="global" type="xs:float"/>
          <xs:element name="intervalo" maxOccurs="unbounded">
           <xs:complexType>
            <xs:simpleContent>
             <xs:extension base="xs:float">
              <xs:attribute name="numero" type="xs:integer" use="required"/>
             </xs:extension>
            </xs:simpleContent>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
         <xs:attribute name="numero" type="xs:integer" use="required"/>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="cantidad_pasajeros_balked" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
```

```
          <xs:element name="piso" maxOccurs="unbounded">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="global" type="xs:integer"/>
             <xs:element name="intervalo" maxOccurs="unbounded">
              <xs:complexType>
               <xs:simpleContent>
                <xs:extension base="xs:integer">
                 <xs:attribute name="numero" type="xs:integer" use="required"/>
                </xs:extension>
               </xs:simpleContent>
              </xs:complexType>
             </xs:element>
            </xs:sequence>
            <xs:attribute name="numero" type="xs:integer" use="required"/>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="cantidad_pasajeros_reneged" minOccurs="0">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="piso" maxOccurs="unbounded">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="global" type="xs:integer"/>
             <xs:element name="intervalo" maxOccurs="unbounded">
              <xs:complexType>
               <xs:simpleContent>
                <xs:extension base="xs:integer">
                 <xs:attribute name="numero" type="xs:integer" use="required"/>
                </xs:extension>
               </xs:simpleContent>
              </xs:complexType>
             </xs:element>
            </xs:sequence>
            <xs:attribute name="numero" type="xs:integer" use="required"/>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
```

element **reporte_edificio/pisos/largo_cola_medio**

diagram



namespace: MODSIM_reporte_simulador_edificio_1.00

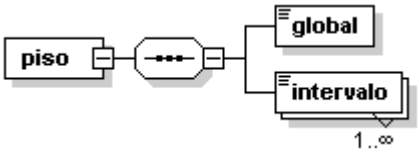
children: **piso**

source

```
<xs:element name="largo_cola_medio">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="piso" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="global" type="xs:float"/>
      <xs:element name="intervalo" maxOccurs="unbounded">
       <xs:complexType>
        <xs:simpleContent>
         <xs:extension base="xs:float">
```

```
                <xs:attribute name="numero" type="xs:integer" use="required"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="numero" type="xs:integer" use="required"/>
    </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
</xs:element>
```

element **reporte_edificio/pisos/largo_cola_medio/piso**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**global intervalo**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:
source	<xs:element name="piso" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="global" type="xs:float"/> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:float"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element> </xs:sequence> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:complexType> </xs:element>

element **reporte_edificio/pisos/largo_cola_medio/piso/global**

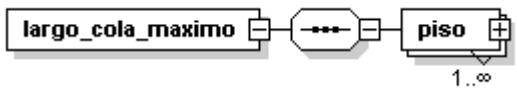
diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	**xs:float**
source	<xs:element name="global" type="xs:float"/>

element **reporte_edificio/pisos/largo_cola_medio/piso/intervalo**

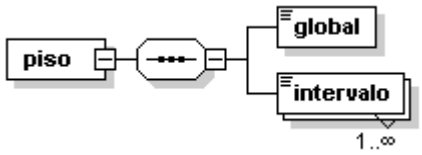
diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00

type	extension of **xs:float**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:
source	<xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:float"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>

element **reporte_edificio/pisos/largo_cola_maximo**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**piso**
source	<xs:element name="largo_cola_maximo"> <xs:complexType> <xs:sequence> <xs:element name="piso" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="global" type="xs:float"/> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:float"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element> </xs:sequence> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </xs:element>

element **reporte_edificio/pisos/largo_cola_maximo/piso**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**global** **intervalo**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:
source	<xs:element name="piso" maxOccurs="unbounded"> <xs:complexType> <xs:sequence>

```
    <xs:element name="global" type="xs:float"/>
    <xs:element name="intervalo" maxOccurs="unbounded">
     <xs:complexType>
      <xs:simpleContent>
       <xs:extension base="xs:float">
        <xs:attribute name="numero" type="xs:integer" use="required"/>
       </xs:extension>
      </xs:simpleContent>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="numero" type="xs:integer" use="required"/>
  </xs:complexType>
</xs:element>
```

element **reporte_edificio/pisos/largo_cola_maximo/piso/global**

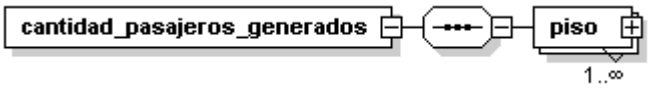
diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	**xs:float**
source	`<xs:element name="global" type="xs:float"/>`

element **reporte_edificio/pisos/largo_cola_maximo/piso/intervalo**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	extension of **xs:float**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:

source	`<xs:element name="intervalo" maxOccurs="unbounded">`
	`<xs:complexType>`
	`<xs:simpleContent>`
	`<xs:extension base="xs:float">`
	`<xs:attribute name="numero" type="xs:integer" use="required"/>`
	`</xs:extension>`
	`</xs:simpleContent>`
	`</xs:complexType>`
	`</xs:element>`

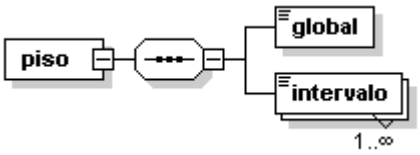
element **reporte_edificio/pisos/cantidad_pasajeros_generados**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**piso**

source	`<xs:element name="cantidad_pasajeros_generados">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="piso" maxOccurs="unbounded">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="global" type="xs:float"/>`

```
          <xs:element name="intervalo" maxOccurs="unbounded">
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base="xs:float">
                  <xs:attribute name="numero" type="xs:integer" use="required"/>
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="numero" type="xs:integer" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **reporte_edificio/pisos/cantidad_pasajeros_generados/piso**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**global intervalo**
attributes	Name: numero; Type: xs:integer; Use: required; Default; Fixed; Annotation
source	<xs:element name="piso" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="global" type="xs:float"/> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:float"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element> </xs:sequence> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:complexType> </xs:element>

element **reporte_edificio/pisos/cantidad_pasajeros_generados/piso/global**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	**xs:float**
source	<xs:element name="global" type="xs:float"/>

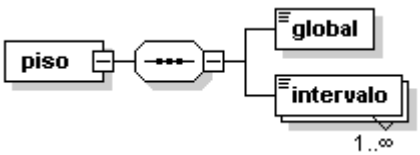
element **reporte_edificio/pisos/cantidad_pasajeros_generados/piso/intervalo**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	extension of **xs:float**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:
source	<xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:float"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>

element **reporte_edificio/pisos/cantidad_pasajeros_balked**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**piso**
source	<xs:element name="cantidad_pasajeros_balked" minOccurs="0"> <xs:complexType> <xs:sequence> <xs:element name="piso" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="global" type="xs:integer"/> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:integer"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element> </xs:sequence> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </xs:element>

element **reporte_edificio/pisos/cantidad_pasajeros_balked/piso**

diagram	

namespace	MODSIM_reporte_simulador_edificio_1.00
children	**global** **intervalo**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:
source	<xs:element name="piso" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="global" type="xs:integer"/> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:integer"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element> </xs:sequence> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:complexType> </xs:element>

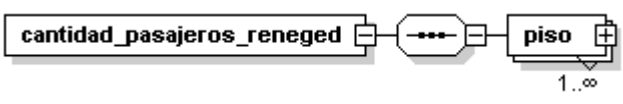
element **reporte_edificio/pisos/cantidad_pasajeros_balked/piso/global**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	**xs:integer**
source	<xs:element name="global" type="xs:integer"/>

element **reporte_edificio/pisos/cantidad_pasajeros_balked/piso/intervalo**

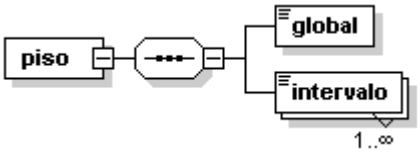
diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	extension of **xs:integer**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:
source	<xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:integer"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>

element **reporte_edificio/pisos/cantidad_pasajeros_reneged**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00

children	**piso**
source	<xs:element name="cantidad_pasajeros_reneged" minOccurs="0"> <xs:complexType> <xs:sequence> <xs:element name="piso" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="global" type="xs:integer"/> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:integer"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element> </xs:sequence> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </xs:element>

element **reporte_edificio/pisos/cantidad_pasajeros_reneged/piso**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
children	**global** **intervalo**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:
source	<xs:element name="piso" maxOccurs="unbounded"> <xs:complexType> <xs:sequence> <xs:element name="global" type="xs:integer"/> <xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:integer"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element> </xs:sequence> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:complexType> </xs:element>

element **reporte_edificio/pisos/cantidad_pasajeros_reneged/piso/global**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00

type	**xs:integer**
source	<xs:element name="global" type="xs:integer"/>

element **reporte_edificio/pisos/cantidad_pasajeros_reneged/piso/intervalo**

diagram	
namespace	MODSIM_reporte_simulador_edificio_1.00
type	extension of **xs:integer**
attributes	Name: numero; Type: xs:integer; Use: required; Default: ; Fixed: ; Annotation:
source	<xs:element name="intervalo" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:integer"> <xs:attribute name="numero" type="xs:integer" use="required"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>