GSPACES, META-MODELO PARA SIMULAR DESALOJOS DE ESPACIOS URBANOS Y ARQUITECTÓNICOS BASADO EN GALATEA

Klaudia Laffaille S.

Trabajo presentado como requisito parcial para optar al título de
MAESTRíA EN MODELADO Y SIMULACIÓN

UNIVERSIDAD DE LOS ANDES
MÉRIDA, VENEZUELA
Mérida, Marzo 2005

UNIVERSIDAD DE LOS ANDES
Consejo de Estudios de Postgrado
Maestría en Simulación

Certificamos que hemos leído y recomendado la aceptación del trabajo de grado titulado "**gSpaces, Meta-Modelo para Simular Desalojos de Espacios Urbanos y Arquitectónicos basado en GALATEA**" presentada por **Klaudia Laffaille S.** como requisito parcial para optar al título de **Maestría en Modelado y Simulación**.

Fecha: Mérida, Marzo 2005

Tutor: _____
Kay Tucci

Jurados: _____
Melín Nava

_____
Jacinto Dávila

Índice general

List of Codes

Agradecimientos

Al equilibrio armonioso del mundo que se encarga de las causalidades de mi vida. Al creador de mi ser, quien me dio la dicha de poseer sus genes y se ha encargado de labrar mi personalidad y hacer de mi quien soy hoy. A la mujer me regaló todo de si para colaborar con mi crecimiento y formación. A las fundaciones de mis principios, mi familia, quienes día a día con sus sonrisas me hacen más fuerte. A esas personas que se han encargado de llenar mi vida de solidaridad y frescura. A quien desde la distancia siempre estuvo para robarme una sonrisa. A mis padres académicos, a esos que siempre confiaron en que podría lograrlo. A quien lee estas líneas por darme la oportunidad de decir lo que necesito decir.

Introducción

Es complicado evaluar que tan adecuada puede ser una propuesta de diseño urbano o arquitectónico, existente o aun no construida, para ser desalojada en caso de que ocurra algún evento que lo requiera. Sin embargo, es importante poder hacerlo ya que gran parte de las ciudades a nivel mundial se encuentran ubicadas en zonas que las convierten en emplazamientos vulnerables y en particular nuestro país, Venezuela, no es la excepción [26] [25]. Esta circunstancia es la principal motivación de esta investigación cuyo objetivo de es construir una herramienta computacional que facilite a sus usuarios la realización de modelos que simulen la dinámica generada por los individuos durante desalojos de espacios urbanos o arquitectónicos.

gSpaces es un meta-modelo basado en el formalismo de simulación de eventos discretos DEVS, que permite realizar una completa descripción de los espacios que se modelan y una evaluación de estados discretos de la interacción de los agentes móviles entre si y con su entorno. gSpaces es llamado meta-modelo por ser un conjunto de herramientas que se utilizan para crear modelos de espacios urbanos y arquitectónicos. La dinámica de los agentes móviles se modela en un nivel discreto, dividiendo el espacio en unidades espaciales o celdas y actualizando las posiciones de loa agentes con una función o regla de movimiento, que puede ser particular para cada espacio debido a que en cada espacio es posible que se desarrollen dinámicas diferentes. Debido a que en un sistema real los individuos pueden no comportarse de las misma manera ante el mismo suceso la regla de movimiento puede ser diseñada para asignar comportamientos diferentes para distintos tipos de agentes dentro del sistema. Las características iniciales del espacio pueden ser generadas a partir de un archivo ASCII en formato .dxf [17] *Data Exchange File* o utilizando una herramienta de gSpaces simplificando el desarrollo de los modelos. Una vez iniciada la simulación la librería permite modificar las características de los espacios que conforman el modelo en tiempo de ejecución con la finalidad de simular eventos que tengan como consecuencia modificaciones en los espacios o los agentes móviles que los habitan.

En esta investigación se crea un meta-modelo para que puedan realizarse modelos de simulación que permitan estudiar situaciones que hasta hace algunos años era imposible conocer, gracias a que la simulación nos permite generar situaciones y sucesos que no pueden modelarse físicamente.

Nuestra librería se presenta en este documento y se realiza su descripción en cuatro capítulos. El primer capítulo explica la principal motivación de nuestro trabajo y las ramas

de investigación que han resuelto sistemas con dinámicas y características similares. El segundo capítulo explica como fue realizada la abstracción de espacios arquitectónicos y urbanos a espacios de nuestra librería y las características de los mismos. El tercer capítulo está dedicado a explicar los detalles particulares de la librería y el cuarto capítulo presenta un conjunto de ejemplos que pueden servir de ejemplos al lector para utilizar nuestra librería.

Nunca pienso en el futuro.
Este llega lo suficientemente rápido.
Albert Einstein

Capítulo 1

¿Planificamos en función de riesgos?

En este capítulo comenzamos con una descripción sobre la vulnerabilidad de nuestras ciudades a nivel mundial, luego comentamos los estudios más relevantes para nuestra investigación sobre simulación de desalojos. Seguidamente explicamos la posibilidad de estudiar y mejorar los diseños arquitectónicos y urbanos para favorecer el desalojo y finalmente planteamos nuestra propuesta y sus aportes con respecto a los estudios realizados anteriormente.

1.1. Ciudades vulnerables

Debido a multiples causas de carácter económicas, históricas, ambientales, estratégicas y de recursos entre otras, gran parte de las ciudades del mundo se encuentran construidas el zonas potencialmente peligrosas donde, existe la posibilidad de que ocurran amenazas naturales, antrópicas o socio-naturales que requieren el desalojo de una edificación o incluso de un sector urbano. De acuerdo con el glosario de términos del Secretariado de Manejo del Medio Ambiente para América Latina y el Caribe [30], "las *amenazas naturales* tienen su origen en la dinámica propia de la corteza terrestre, de la atmósfera y de la biota como terremotos, erupciones volcánicas, huracanes, tsunamis, lluvias torrenciales, epidemias, etc. *Las amenazas antrópicas* son atribuibles a la acción humana directa sobre elementos de la naturaleza y/o de la sociedad como vertimiento de residuos sólidos o efluentes, que provoca contaminación del agua; liberación de partículas contaminantes al aire, que ocasiona enfermedades respiratorias; muertes por la guerra, etc y las *amenazas socio-naturales* se expresan a través de fenómenos de la naturaleza, pero en su ocurrencia o intensidad interviene la acción humana como inundaciones por degradación de riberas, incremento de la escorrentía por urbanización, deficit de agua potable debido a la contaminación de acuíferos, etc."

La vulnerabilidad de los asentamientos urbanos y sus edificaciones a estas amenazas hace necesario el estudio cuidadoso del emplazamiento, el diseño y la construcción de ciudades y edificaciones con la finalidad de que puedan considerarse relativamente seguras o menos vulnerables.

La **Vulnerabilidad** se define como "predisposición intrínseca de un elemento expuesto a ser afectado por la ocurrencia de un evento de cierta intensidad"[29].

A través de la historia el hombre se dedicó a escoger los lugares de emplazamiento para sus ciudades de acuerdo con factores como: estrategias económicas, proximidad a cuerpos de agua, clima, vegetación, belleza de los parajes, estrategias militares, etc.

La principal razón por la que nuestras ciudades actuales son vulnerables a amenazas naturales, antrópicas o socio-naturales es que no fueron planificadas para ellas; las ciudades hispanoamericanas por ejemplo, fueron fundadas a raíz de un fenómeno geopolítico que perseguía la expansión y la incorporación del nuevo territorio al imperio español como lo explica más detalladamente M Teresa Padilla Aguilar en su artículo "Como aparecieron las ciudades del nuevo mundo"[28]; el Rey Carlos V creó las "Leyes de Indiasçon la finalidad de especificar a los 'descubridores' como son llamados en las propias leyes, como debían ser creadas las ciudades [27]. Así como las ciudades son planificadas con unas determinantes particulares de la época, con las edificaciones ocurre lo mismo, el uso, la estética y las tecnologías constructivas existentes; la economía del lugar, y la disponibilidad de espacio son las determinantes comúnmente manejadas al diseñar una edificación, por lo tanto el diseño de las mismas responde al entorno social y económico; a la funcionalidad y las normas vigentes para la época en que fueron diseñadas, épocas en las que disminuir la vulnerabilidad no era una de las prioridades. Con el paso de los años muchas de estas construcciones demuestran poseer errores de configuración formal, estructural y de ubicación, que no se contemplaban para el momento en que fueron construidas. Además, el entorno va cambiando a medida que aumenta la densidad y un lugar de emplazamiento que pudo haber sido seguro en determinada época actualmente puede ser un sector vulnerable. Actualmente, la disminución del la vulnerabilidad es una condicionante mas de diseño y existen normas que intentan garantizar ciertas condiciones para disminuir la posibilidad de que sean creadas edificaciones en sectores peligrosos y con problemas de configuración formal y estructural pero, la evasión de las normas, la auto-construcción y la invasión han dejado de lado estos criterios para crear ranchos, barriadas y edificaciones sin los estudios previos necesarios, que son descartados por necesidades que parecen ser más importantes como la inminente necesidad de un lugar para vivir o el interés de invertir obteniendo el máximo de ganancias. Otra de las dificultades que se suma al problema es que es difícil probar si una edificación o un sector urbano podrá ser desalojado eficientemente cuando aún no ha sido construido y preguntas como: ¿Qué ocurre cuando las personas necesitan desalojar una edificación o un sector urbano? ¿Están realmente nuestras edificaciones y ciudades diseñadas para ser desalojadas? ¿Funciona adecuadamente el diseño para facilitar el desalojo de la edificación o del sector?, por lo general no pueden ser respondidas con facilidad.

Con la finalidad de colaborar con el estudio de la esta dinámica y el planteamiento de

posibles respuestas a estas y otras preguntas, en nuestro trabajo se desarrolló un metamodelo computacional que sirve como herramienta para elaborar modelos que simulen la dinámica generada por individuos que desalojan espacios construidos e incluso dinámicas en espacios aún no construidos.

1.2. Estudios realizados en el tema

El interés sobre la funcionalidad de los diseños arquitectónicos en caso de desalojos no es particular de nuestra investigación, anteriormente se han realizado varios estudios sobre el tema utilizando diferentes tipos de modelos que responden a diferentes ramas investigativas.

En el año 2000, Helbing, Farkas y Vicseck [22] construyeron un modelo que simula las características de la dinámica propia del pánico de un desalojo. Para simular el movimiento de las personas Helbing et al. [22] utilizan funciones de transición que hacen variar para cada iteración de tiempo la posición del peatón y su dirección. Para esto plantean que las personas se mueven bajo la acción de fuerzas de diferente tipo, una fuerza resultante compuesta por dos fuerzas la producida por la interacción de los individuos con las paredes y la generada por la interacción con los demás individuos. De esta manera las personas se comportan como partículas que son influenciadas por fuerzas externas que condicionan su movimiento. Esta regla de movimiento está basada en una actualización de la velocidad de desplazamiento resultante de la interacción de la partícula con la fuerzas que en ella inciden. Posteriormente, en el año 2003 *Escobar y de la Rosa* [16] realizan una investigación basándose en el modelo de Helbing et al. y agregan una nueva fuerza que influye sobre las partículas, la interacción con obstáculos. Este tipo de modelos permite conocer la dinámica del fenómeno en general sin entrar en detalles de cierto tipo de características como, por ejemplo, el volumen que ocupan las partículas.

Otra manera de modelar este tipo de fenómenos de naturaleza continua es discretizando mediante el uso de técnicas de autómatas celulares. Los autómatas celulares que permiten realizar análisis numéricos para comprender, por medio de un modelo discreto, el comportamiento de sistemas analógicos. *Kirchner y Schadschneider* [9] en el 2002 desarrollaron un automata celular con el propósito de simular el mismo problema. Basaron su modelo en la biónica, describiendo la interacción entre las personas usando ideas basadas en la quimiotaxis. La quimiotaxis es un proceso químico que utilizan las células para informar a otras que deben dirigirse hacia un lugar determinado [13]. En el modelo desarrollado por Kirchner y Schadschneider. los individuos guían a otros individuos hacia las salidas. Estos modelos son muy eficientes para simular comportamientos en masa y la influencia que produce en un determinado individuo el comportamiento de los demás o otros individuos.

1.3. Estudiar y mejorar los diseños arquitectónicos y urbanos para favorecer el desalojo

Es posible incluir como una condicionante de diseño la efectividad del desalojo de un determinado espacio en caso de un evento catastrófico. Sin embargo el diseñador puede decidir si considera dicha condicionante y como hacerlo, que factores considerar y a que dar prioridad en el momento de concebir la propuesta de diseño. Para cada país existen normas que indican algunas de las cualidades que una edificación debe cumplir con respecto a las salidas de emergencia además de la configuración formal y estructural pero en mucho de los casos sólo regulan pocas variables como, por ejemplo, el tamaño de las salidas y la distancia de estas hacia el punto más alejado de la edificación. Si el diseño de una edificación se encuentra más comprometido con dicha condicionante de diseño es porque existe un interés particular del diseñador. Además, una vez que se ha elaborado una propuesta de diseño es conveniente poder estudiar si realmente la decisiones tomadas y las cualidades del diseño mejoran significativamente la fluidez y velocidad del desalojo. La meta de incorporar la efectividad de los desalojos como condicionante de diseño es que los espacios una vez construidos funcionen adecuadamente. Debido a esto se hace necesario modelar las situaciones catastróficas y los espacios para poder realizar diferentes estudios referentes a la configuración como la configuración formal de los espacios, la ubicación de las salidas con respecto a la forma del espacio contenido, la ubicación de zonas seguras dentro del espacio, la forma y tamaño de las vías de desalojo, etc.

El conjunto de variables citadas anteriormente pueden cambiarse creando diferentes propuestas de diseño pero, es necesario poder probar y comparar diferentes alternativas. Además, las áreas destinadas para circulación en un espacio son determinadas y planificadas por el diseñador. Es entonces, en el proceso de diseño, cuando se deciden que lugares serán destinados como recorridos de circulación y de desalojo. Sin embargo la mayor parte de las decisiones tomadas al respecto están basadas en suposiciones acerca de como se comportarán las personas y que proceso puede ser el más adecuado para realizar el desalojo del espacio.

1.4. Planteamiento de la propuesta

Con la finalidad de facilitar el modelado de espacios arquitectónicos y urbanos y la evaluación de su efectividad en caso de desalojos,en esta investigación se desarrolló, diseñó e implementó un ***Meta-Modelo Computacional*** **para Evaluar el Rendimiento de Diseños Urbanos o Arquitectónicos para Efectos de Desalojos de Personas en Caso de Eventos Catastróficos.**

Un meta-modelo "es un modelo de cómo hacer un modelo, es una descripción del cómo hacer una descripción. En los lenguajes computacionales como Java y GALATEA, un meta-modelo puede adoptar la forma de una plantilla de texto que se completa con estructuras lingüísticas particulares para producir cierto modelo. Antes de completarla, esa plantilla es

un meta-modelo. La plantilla también puede ser vista como un patrón para una familia de modelos, por exactamente las mismas razones"[32].

En nuestra investigación se desarrolló un meta-modelo espacial que sirve como patrón general para toda una familia de modelos de espacios urbanos o arquitectónicos, que pueden ser sometidos a cambios morfológicos y estructurales durante la simulación e incluyen a los habitantes de esos espacios y a sus dinámicas.

Con la finalidad de seleccionar la manera más adecuada para simular las condiciones propias de un espacio urbano o arquitectónico y la dinámica de las personas que lo habitan cuando realizan un desalojo, se analizaron las diferentes formas de modelar el problema que nos compete citadas anteriormente y en (véase sección 1.2), se compararon en busca de las cualidades que debería poseer una solución eficiente para realizar un meta-modelo. Nuestra librería, gSpaces, les permite a los modelistas abordar la complejidad de modelado espacial, simplificando la creación de modelos particulares de espacios urbanos, sin que deban preocuparse por todos los detalles de implementación.

Los modelos con autómatas celulares [1, 5, 8] consisten en una o varias retículas superpuestas que suelen denominarse capas o *layers*. El estado de una celda lo conforman el conjunto de valores que las capas tiene en la posición de la celda. Cada una de las celdas, pertenecientes al autómata, actualiza su estado en un intervalo de tiempo seleccionado por el modelista y dicha actualización es realizada una vez que han sido recorridas todas las celdas del autómata. Modelar una regla de actualización de estado para un automata celular consiste en una actualización de estado que se realiza en cada una de las celdas en función de su estado y el de las celdas vecinas.

Estas características hacen que los modelos de sistemas espacialmente extendidos elaborados con autómatas celulares sean simples y claros de realizar. Sin embargo, en el caso de simular el movimiento de personas poseen las siguientes desventajas:

- *Exclusión y eventos simultáneos*: En los modelos en los que la actualización del estado de cada una de las celdas se realiza una vez que todas las celdas han sido recorridas, se presenta la posibilidad de que dos individuos puedan ocasionar el cambio de estado de una celda simultáneamente. Para evitar esto es necesario evitar que dos individuos puedan encontrarse en una misma celda o tolerar que las celdas tengan capacidad infinita. La solución que resuelve este problema particular, evitando que dos individuos puedan habitar la misma celda en el mismo instante de tiempo, se le conoce como exclusión. Por otra parte, este tipo de actualización hace que dos eventos simultáneos no sean diferenciables.

- *El barrido de las celdas*: De acuerdo a como se realice el barrido de las celdas para realizar la actualización de estados puede que unas celdas posean prioridad en el momento de acceder a una celda vacía o a una salida.

- *Rigidez del espacio o granularidad*: La forma de los límites físicos que envuelven un espacio deberá adaptarse a la reticula con la que se realiza el modelado, por lo tanto una pared inclinada consistirá en un conjunto de celdas con determinado estado que

se acercan a modelar la forma del cerramiento pero seguirán siendo celdas alineadas de acuerdo a la inclinación en el plano del cerramiento.

- *Individuos no diferenciables*: En un modelo de autómatas celulares no existe realmente un objeto que se desplace a través de las celdas, sólo ocurren cambios de estado en las celdas, por esto no es posible seguir el recorrido de manera detallada de una o varias de las personas dentro de la simulación.

Por otra parte, **los modelos realizados con ecuaciones diferenciales** [16, 22] modelan el espacio y el tiempo como variables continuas, la actualización de sus variables se realiza por medio de ecuaciones que son calculadas en intervalos lo suficientemente cortos como para que no se dejen de registrar eventos ocurridos durante de la simulación. Esta una de sus ventajas cuando el problema real a simular posee como características espacio y tiempo continuos. Los modelos de movimiento elaborados por medio de ecuaciones diferenciales son similares a los utilizados en hidrodinámica o dinámica molecular, donde se asume que el fluido es un continuo. Al ser utilizados para modelar desalojos poseen las siguientes desventajas:

- *Humanos como partículas de un fluido*: Al proponer que el movimiento de personas puede resolverse por medio de ecuaciones diferenciales se explica como un conjunto, un fluido que obedece a ciertas reglas descritas por medio de las ecuaciones propuestas, pero, en un conjunto de individuos que se desplazan a través de un espacio es posible que algunos de ellos no sigan el comportamiento de la mayoría, e incluso podría ocurrir que existan tantos comportamientos diferentes como individuos.

- *Complejidad del modelo y su resolución*: Las ecuaciones que definen un modelo de este tipo, a pesar de ser soluciones claramente demostrables matemáticamente, son complejas, existiendo una gran dificultad para comprender, plantear y resolver problemas por medio de esta forma de modelado. Debido a lo expuesto anteriormente un modelo de ecuaciones diferenciales es conveniente que sea planteado cuando dicha complejidad se ve justificada por el tipo de respuestas que se desea obtener del modelo, en nuestro caso particular, donde se desea brindar la posibilidad de que cada agente móvil posea un comportamiento diferentes esta complejidad no puede calificarse como necesaria.

- *Dificultad para plantear las condiciones de borde*: El espacio está definido por medio de fórmulas que explican un gran espacio continuo, los límites del espacio a simular se modelan como condiciones de borde. Plantear las condiciones de borde para dicha formula es complejo debido al número de variables que se manejan. En nuestros caso particular la dificultad aumenta porque es posible que el espacio que se desee definir posea una geometría compleja y se desea ofrecer la posibilidad de modificarlo durante la simulación, esto implica en un modelo de ecuaciones diferenciales calcular nuevos límites o bordes cada vez que el espacio cambie su forma circunstancia que

hace que este tipo de modelos sean poco prácticos para modelar nuestro problema en particular.

- *Individuos no diferenciables*: Como se mencionó anteriormente, los modelos explicados por medio de ecuaciones diferenciales no permiten tratar a cada persona como individualidad, no es posible determinar exactamente cual fue el recorrido de uno de los individuos durante la simulación debido a que al ser considerado como un fluido se trata a todas las personas como 'partículas' que se rigen por determinadas reglas sin poder diferenciarlas unas de otras.

Una vez estudiadas las características de las formas de modelado utilizadas anteriormente, para resolver el problema que nos compete se incorporaron cualidades de ambas formas de modelado creando un meta-modelo híbrido discreto en tiempo y continuo en espacio con las siguientes características :

- *El espacio* es modelado como un plano definido por dos ejes de coordenadas X y Y y líneas que contiene y definen sus límites. Al ser un plano continuo, cada individuo posee una posición real, definida por un par de numeros reales para los ejes X y Y y un vector de velocidad continuo. Sin embargo, para simplificar las reglas de movimiento, disminuir la cantidad de cálculos que se realizan en cada actualización del modelo y mejorar la velocidad de computo, algunas variables son almacenadas como atributos de un objeto que divide el espacio en **sectores o celdas .** Las celdas pueden brindar a cada uno de los individuos el valor de ciertas variables, es decir, cada una los conoce y todo individuo que vaya a actualizar su posición en un instante de tiempo determinado puede conocerlas y en algunos casos modificarlas durante su ejecución de la regla de movimiento.

  De esta manera, el espacio sigue siendo un plano continuo y para cada sector, cuyo tamaño es seleccionado por el modelista, están definidas las variables que la celda puede brindar.

  Los límites del espacio están definidos por objetos descritos espacialmente como líneas que pueden interceptar el vector de desplazamiento planificado por los individuos al momento de actualizar su posición. Dichos objetos pueden ser de dos tipos, **puertas** o **paredes**. Las puertas, definidas por el segmento contenido entre dos puntos, pueden ser entrada o salida del espacio incorporando individuos a un espacio o enviándolos a la salida general del modelo y planificando su salida del espacio actual. Las paredes también están definidas por el segmento comprendido entre dos puntos y es el espacio quien determina si un individuo deberá interrumpir su recorrido para no colisionar con una pared sugiriéndole una posición posible o correcta. El espacio determina si una posición propuesta por un individuo para el siguiente instante de tiempo es posible cuando durante su recorrido no intercepta paredes y se mantiene dentro de los límites del espacio. La única forma de que un individuo pueda tener una posición que se encuentre fuera del espacio que lo contiene es que atraviese una puerta abierta que

se encuentra desocupada. Esta manera de modelar el espacio hace posible que la velocidad de un individuo sea continua, dentro de un espacio continuo sin modificaciones en el sistema de referencia espacial cada vez que un individuo cambia de **unidad espacial o celda**. Por otra parte la discretización de algunas variables requeridas para la actualización de posiciones mejora la velocidad de computo y simplifica las reglas de movimiento siendo más claras para el modelista a pesar de tratarse de un espacio continuo. Las cualidades propias del espacio y sus límites permite modelar espacios con formas irregulares sin implicar mayores complicaciones para el modelista con respecto a la definición de los límites o *condiciones de borde* del modelo.

- *El tiempo* es una variable discreta que permite actualizar el estado del modelo en instantes de tiempo definidos por el *paso de simulación* que es seleccionado por el modelista. Entre dos actualizaciones del modelo las velocidades de las personas, en principio, son constantes y por lo tanto las personas se moverán en línea recta a menos de que su vector velocidad intercepte una pared o atraviesen una puerta. Esta condición hace necesario que el tiempo que transcurra entre cada actualización de las posiciones debe estar de acuerdo con la precisión y el nivel de detalle que se requiera en el modelo y poseer una relación correcta con respecto al sistema si se desea obtener resultados que puedan responder a interrogantes plateadas para el sistema real. El paso de simulación puede ser tan pequeño como sea necesario, llegando a poseer cualidades similares a las interacciones realizadas en un modelo de ecuaciones diferenciales si el caso lo requiere. Por otra parte, tener instantes separados de tiempo para actualizar el estado del modelo permite simular decisiones discretas que pueden cambiar para el instante de tiempo siguiente en función del nuevo estado del sistema. La actualización del modelo en tiempo discreto permite aplicar la teoría sobre el comportamiento humano que se utiliza en inteligencia artificial donde, se plantea que un ser inteligente basa sus decisiones en la información que obtiene del ambiente [19]. Por otra parte, una regla de movimiento que actúa en tiempos discretos se basa en decisiones discretas, que pueden ser más sencillas de programar y entender que las ecuaciones diferenciales de los modelos continuos, librando de complejidad la programación y planeación del modelo.

- *Los individuos* son simulados como entidades independientes u objetos. Cada individuo actúa como una entidad que puede tener un comportamiento diferente con respecto a los otros individuos pertenecientes al mismo espacio o modelo. De esta manera es posible que dentro de un mismo espacio existan individuos que se comporten de manera diferente al resto como podría hacerlo una persona herida, un rescatísta o alguien que se encuentra paralizado por el pánico.

- *La actualización del estado del modelo* se realiza durante cada iteración de tiempo para cada uno de los espacios que conforman el modelo. Cada espacio recorre aleatoriamente las celdas ocupadas que lo conforman actualizando instantáneamente la posición de cada uno de los individuos de acuerdo a la función de movimiento que

se le haya asignado. La actualización concluye una vez que todos los espacios han asignado las posiciones de los individuos que albergan. El recorrido aleatorio de las celdas permite que no existan prioridades entre los sectores que componen el espacio permitiendo que todos los individuos que puedan dirigirse hacia un mismo punto en un instante de tiempo determinado tengan la misma posibilidad de hacerlo.

El meta-modelo está desarrollado como un módulo de la plataforma de simulación GALATEA. "GALATEA es una plataforma de simulación de sistemas multi-agentes. Es una colección de software para simular, sobre modelos escritos en lenguajes de diversa naturaleza: procedimental, orientada a los objetos, dinámica de sistemas y orientación a los agentes. GALATEA es descendiente de GLIDER, un simulador de eventos discretos (DEVS) desarrollado en la Universidad de Los Andes, Venezuela, a finales del siglo XX. GALATEA extiende a GLIDER incluyendo en la semántica DEVS las abstracciones necesarias para describir AGENTES: entidades que perciben un ambiente, razonan sobre lo que ven y actúan sobre el ambiente, tratando de cumplir sus propios objetivos"[32]. GALATEA fue seleccionada como plataforma para crear el meta-modelo desarrollado en nuestra investigación debido a que es un ambiente de simulación orientado a objetos que soporta simulación DEVS que puede ser usada para modelar sistemas continuos reduciendo la simulación continua a un discreto por medio de un paso de simulación muy 'pequeño'. Por otra parte, GALATEA permite diseñar extensiones, como este meta-modelo, que será incorporado a la colección de software que la conforma.

Nuestro cuerpo y nuestra percepción siempre nos requieren a aceptar como centro del mundo aquel medio ambiente con que nos rodea.
Merleau-Ponty.

Capítulo 2

Modelando espacios en gSpaces

Este capítulo lo dedicaremos a establecer un concepto general de espacio y explicar como es modelado por nuestra librería. Comenzaremos describiendo las características generales de el meta-modelo que creamos. Posteriormente estudiamos los conceptos generales de espacio, espacio arquitectónico y espacio urbano para establecer un concepto general de espacio que nos servirá al momento de modelar nuestra librería. Finalmente se explica de manera metodológica, como se realiza la abstracción necesaria para modelar con gSpaces espacios que pertenecen al mundo real.

2.1. Definición general de espacio, espacio arquitectónico, espacio urbano y el espacio del modelo

Uno de los objetivos de esta investigación es que gSpaces sirva como herramienta para modelar espacios arquitectónicos y urbanos, que contengan agentes móviles con diversas dinámicas. Con esta finalidad en mente se estudiaron varios conceptos de espacio seleccionando las características que son necesarias para simular un espacio y las posibles dinámicas de los individuos que contiene. Sin embargo es importante destacar que cada disciplina puede utilizar un concepto de espacio que se adapta al sistema en estudio, por ejemplo, las siguientes definiciones son relativas a la disciplina o rama investigativa que las define:

- Extensión indefinida, medio sin limites que contienen todas las extensiones finitas. Parte de esta extensión que ocupa cada cuerpo.
- Distancia entre dos o más objetos.

- Cada una de las partes que componen un programa radiofónico o de televisión.
- Transcurso de tiempo, hablar por espacio de una hora.
- Estructura que nos asegura que al componer dos elementos pertenecientes al espacio (elementos a los que llamaremos vectores) de acuerdo a una cierta operación, el resultado sigue siendo un elemento del espacio. En otras palabras, la suma de vectores será un vector y no cualquier otra cosa, como podría ser un punto.
- Distancia o el área entre o alrededor de las cosas.

Según Christian Norberg Schulz, en su libro, Existencia, Espacio y Arquitectura [6] existen las siguientes definiciones de espacio, orden ascendente de abstracción:

- El espacio *pragmático*, de acción física, el espacio en el que el hombre actúa, el concepto que integra al hombre con su ambiente orgánico.
- El espacio *perceptivo*, de orientación inmediata, es el espacio que el hombre percibe, es esencial para su identidad como persona.
- El espacio *existencial*, que forma para el hombre la imagen estable del ambiente que le rodea, le hace pertenecer a una totalidad social y cultural.
- El espacio *cognoscitivo del mundo físico*, es un concepto que implica pensar acerca del espacio.
- El espacio *expresivo o artístico*, es el espacio creado por el hombre para expresar su imagen del mundo. El espacio arquitectónico es un espacio expresivo, y como todo espacio expresivo, su creación es tarea de personas especializadas, constructores, arquitectos y planificadores.
- El espacio *estético*, es la construcción abstracta que sistematiza las propiedades de los posibles espacios expresivos. El espacio estético es estudiado por teóricos en arquitectura y filósofos.
- El espacio *lógico*, es el espacio abstracto de las relaciones lógicas, que ofrece el instrumento para describir los otros espacios.

En nuestro caso, el tipo de espacio que se desea modelar es el que tiene la capacidad de contener individuos en movimiento y más específicamente, espacios que pueden ser habitados por personas. Una vez que precisamos que los espacios estarán habitados por personas es necesario definir conceptos para *espacio arquitectónico y espacio urbano*, contenedores de personas importantes en los asentamientos humanos actualmente.

El **espacio arquitectónico** es el elemento primordial de la Arquitectura, al que ella delimita y pormenoriza delimitado por el volumen. A pesar que el espacio se encuentra definido materialmente por el volumen no siempre coincide con la forma material que lo

delimita, pudiendo variar mediante niveles interiores, color y texturas[21]. Los espacios arquitectónicos pueden colindar unos con otros y comunicarse cuando en dos de ellos existe un sector abierto que permite atravesarlos simultáneamente a través de la intercepción de sus límites. Como se explicaba brevemente en el párrafo anterior, los límites de un espacio arquitectónico pueden estar definidos de diversas maneras, pueden ser volúmenes físicos, cerramientos (puertas, paredes, celosias, etc), cambios de uso con respeto a otros espacios, la textura de los suelos, diferencias de nivel, luces, sombras etc. Como espacio estético [6] existen diversas formas de limitar y definir los espacios que son definidas por personas especializadas en la disciplina.

El **espacio urbano** es el elemento contenido en las áreas públicas y semi públicas de una ciudad, una suma de los espacios requeridos por la dinámica de una ciudad que en muchos casos posee como límite superior el cielo. Se desarrolla dentro del flujo de los individuos que participan de él y muta según los cambios de las dinámicas que estos generan. No dista mucho del concepto de espacio arquitectónico pero deja de estar contenido dentro de una edificación para ser el producto de un conjunto de volúmenes arquitectónicos y cambios de uso que lo delimitan.

Puede decirse entonces que tanto los espacios urbanos como arquitectónicos tienen como características comunes el ser ***volúmenes contenidos dentro de límites tangibles o perceptibles para los individuos que los habitan y desarrollan una dinámica dentro de ellos.*** Los límites de ambos espacios pueden ser de diversos tipos como se mencionaba anteriormente y, están definidos en función de los individuos que los habitan y sus respectivas percepciones. Los espacios donde interactúan individuos son extensiones con características particulares que la diferencian de otras de su misma clase, sus límites son conocidos (incluso si es infinito) y poseen la capacidad de contener a otras entidades y espacios.

Christian Norberg Schulz plantea un concepto de espacio que llama existencial que define de la siguiente manera "sistema relativamente estable de esquemas perceptivos o imágenes del ambiente circundante. Siendo una generalización abstraída de las similaridades de muchos fenómenos, ese espacio existencial tiene "carácter objetivo". El autor propone que las idea de un mundo estructurado se desarrolla gradualmente durante la infancia, y que necesariamente, comprende un desarrollo de nociones espaciales. Durante este proceso de aprendizaje cada ser humano desarrolla el conjunto de relaciones y criterios que le permiten entender y codificar un "sistema de lugares [2]explica que "El desarrollo del concepto de lugar y del espacio como un sistema de lugares es, por consiguiente, una condición necesaria para hallar un sitio firme donde hacer pie existencialmente". Será en función de esta organización existencial desarrollada por los habitantes de los espacios lo que determine la percepción que poseerán de quien los contiene. Sin embargo, a pesar de que las percepciones de espacio pueden desarrollarse en contextos muy variados existen esquemas elementales de organización comunes y Norberg Schulz explica que "consisten en el establecimiento de centros o lugares (proximidad), direcciones o caminos (continuidad) y áreas o regiones (cerramientos o cercados)". El concepto de espacio y de las dinámicas que se desarrollan dentro de los mismos en el momento en que son habitados, utilizados

en nuestra librería, se apoyan en las definiciones realizadas por Norberg Schulz sobre *centros, direcciones o caminos* y *áreas o regiones.* Dichos conceptos fueron aplicados como cualidades o características propias del modelado del sistema.

Los **centros o lugares** tienen que ver con la manera como el ser humano percibe el espacio; al encontrarnos en una determinada posición, subjetivamente somos el el centro de este espacio y cuando desarrollamos esquemas espaciales, la noción de centro está establecida no sólo como un medio de organización general, sino que ciertos centros están situados externamente como puntos de referencia en el ambiente circundante. Nuestro meta-modelo incorpora este concepto creando un modelo de espacio que permite a los agentes tener una percepción directamente relacionada con su posición. Los agentes, conocen las características de su contexto inmediato observando a su alrededor y calificando, con esta acción, a su posición como centro simbólico del espacio en el momento de la observación. Las cualidades del espacio que obtendrá como distancias, cantidad de agentes que lo rodean y otras, están referidas a su contexto inmediato y a su posición actual, por ejemplo, la distancias están calculadas de acuerdo a al centro de la pequeña porción de espacio que lo rodea; más adelante explicaremos este concepto con más detalle cuando hablemos de *espacio semi-continuo* en la sección 3.1.

Norberg Schulz explica que toda **dirección o camino** se caracteriza por su continuidad, en tanto que el lugar está determinado por la proximidad de sus elementos definitorios o límites y, eventualmente por su cerramiento. El camino es concebido como una sucesión lineal, hay una dirección lineal a seguir hacia una meta que hay que alcanzar, pero durante el recorrido ocurren acontecimientos y el camino también posee un carácter propio. Lo que ocurre a lo largo del camino, se agrega a la tensión creada por la meta que hay que alcanzar y el punto de partida dejado atrás. En ciertos casos el camino desempeña la función de ser un eje organizador de los elementos que lo acompañan. gSpaces incorpora este concepto en la manera como son actualizadas las posiciones de los agentes móviles, cada agente posee un camino o dirección que desea seguir, un punto hacia en que desea llegar y en función de esto realiza sus desplazamientos. Cada agente realiza un desplazamiento de acuerdo al punto hacia el que se dirige, el tiempo que posee para realizarlo y su velocidad. De esta manera estructura su interacción con el espacio de acuerdo al recorrido que desea realizar, la proximidad de los límites o elementos definitorios del espacio, las salidas y cerramientos. Cada agente puede poseer una dirección diferente a la del resto de agentes con los que comparte el espacio y el modelista puede especificar cual será la meta o criterio que seguirá cada uno de ellos para alcanzar la siguiente posición como se explicará más adelante (véase 4).

Los caminos dividen y estructuran las zonas que rodean al hombre en áreas más o menos conocidas, a las áreas más conocidas Norberg Schulz les llama **áreas o regiones**, región implica una definición "por su cerramiento o por la proximidad y semejanza de los elementos constituyentes". Pero difiere de aquel en que "nuestra imagen de los alrededores comprende áreas a las que nosotros no pertenecemos y que no tienen función de metas". Norberg Schulz concluye este apartado diciendo que "la imagen que el hombre tiene de las regiones está influida por factores físicos y funcionales, así como sociales y culturales, esto

es, por los objetos básicos de que dispone para su orientación". Los espacios pertenecientes a nuestro meta-modelo son definidos como las regiones explicadas por el autor. Cada espacio posee un conjunto de límites que lo definen y delimitan, son creados por el modelista en función de la percepción del espacio y las metas que tendrán los agentes móviles que estarán en el. Dichos límites son definidos por factores físicos, funcionales, sociales, perceptivos, entre otros.

Una vez incorporados estos conceptos al modelado del sistema, propios de los espacios y las dinámicas que se desarrollan dentro de ellos al ser habitados, podemos comenzar el proceso de modelado del espacio arquitectónico o urbano descomponiéndolo en cada uno de los espacios que lo conforman. A cada uno de ellos lo asociamos con un `Space`, que es un pieza de software que representa un espacio cartesiano bidimensional, es decir que la posición de cada uno de los objetos que se encuentran en él viene definida por un par de coordenadas (x, y), dadas en las unidades de medida que el modelista le asignó a ese `Space`.

Los **límites** de un `Space` están definidos por medio de dos tipos de objetos, los que pueden ser atravesados por los agentes móviles que llamaremos (`Doors`) y los que no pueden ser atravesados, llamados (`Walls`). Los primeros, `Doors` son 'permeables' y permiten que los agentes móviles que se encuentran en un determinado espacio puedan salir de éste e ingresar a otro. Cada `Door` posee una capacidad que debe ser asignada por el modelista. Esta cantidad determina cuántas de personas pueden atravesarlo en un mismo instante de tiempo. La capacidad puede variar en tiempo de ejecución e incluso puede bloquearse, impidiendo que los individuos puedan 'pasar' a través del `Door`. Además, poseen un *índice de selección* que el modelista puede asignar y usar para indicar la posibilidad de que un agente móvil la seleccione como *salida* del `Space`. El segundo tipo de límite son los `Walls` que no son 'permeables'. Tanto los `Walls` como las puertas están definidos dentro del plano del espacio como segmentos de recta comprendidos entre dos puntos. Cada `Space` debe poseer un conjunto de `Walls` y `Doors` que lo limitan y contienen.

Cada `Space` está dividido en **celdas**, que son espacios cuadrados cuya longitud de lado es una unidad definida por el modelista al asignar la resolución del `Space`. Las celdas poseen la cualidad de conocer un conjunto de variables que pueden ser útiles para el modelista al momento de programar la regla de movimiento utilizada para actualizar la posición de cada agente móvil. Entre las variables discretizadas por las celdas tenemos, la lista de `Walls`, ordenada menor a mayor según la distancia a la que están de la celda; la lista de (`Doors`) ordenada según su índice de selección y la distancia a la celda; la dirección hacia la que se dirigió el último agente móvil; la cantidad de personas; el índice de obstáculos; y la dirección preferencial de movimiento. Todas estas variables son relativas a cada celda y las distancias son medidas desde el centro de la celda. El índice de obstáculos es un valor que puede ser utilizado por el modelista para retrasar el avance de los agentes móviles dentro de una celda pero todo dependerá de lo que se programe en la regla de movimiento para el espacio. En función del tipo de modelo, el modelista debe considerar si es relevante evaluar si dos individuos están ocupando exactamente el mismo espacio porque de no serlo, es recomendable que las celdas posean un área mayor a la que ocupará una persona pero lo suficientemente pequeña como para que no pueda contener a todo el `Space`. Si el tamaño

de las celdas es igual al que ocuparía una persona en un espacio real, el modelista debe contemplar en fenómeno de exclusión al modelar las reglas de movimiento (véase secciones 4.1 y 3.3).

La **actualización de las posiciones** es realizada por cada gSpaces para cada interacción de tiempo. El espacio recorre su lista de celdas aleatoriamente y actualiza las posiciones de cada uno de los individuos que se encuentran en las celdas ocupadas de acuerdo a la regla de movimiento que el modelista le haya asignado (véase también sección 4). Pueden asignarse funciones de movimiento diferentes para cada espacio e incluso dentro de estas considerar diferentes reglas de movimiento para cada tipo de persona.

Actualmente, la diferencia de nivel entre dos espacios es considerada un factor que divide o separa espacios que se comunican físicamente. Por ejemplo la relación existente entre la planta baja de una edificación y su primer piso puede ser modelada como dos espacios que se comunican por un límite permeable `Door` ubicado en un lugar determinado, de manera que los individuos que se encuentran en el piso 1, al atravesar determinada `Door` ingresarán al espacio que representa la planta baja de la edificación con un retraso asignado por el modelista. También es posible, si se desea estudiar la dinámica que se produce en el medio de comunicación que en nuestro ejemplo se trata de una escalera, se pueden modelar tres espacios diferentes comunicados por `Doors`, la planta baja, la escalera y el piso 1.

2.2. Abstracciones, como modelar un espacio 'real' con gSpaces

Como ya se ha explicado en la sección 2.1, los límites de un espacio arquitectónico o urbano están definidos en función de la percepción de las personas que lo habitan y esta es la razón por la cual los límites de un `Space` serán definidos por el modelista en función de la percepción que los agentes móviles tendrán del espacio que ocupan. En el proceso de definir los límites de un espacio habrá que determinar que sectores del mismo son permeables, es decir, los que poseen la capacidad de ser atravesados por las personas y cuales no lo son, por ejemplo los límites de un espacio pueden ser:

- Los objetos que separan físicamente espacios contiguos e impiden el desplazamiento de los individuos desde uno hacia otro como cerramientos, cambios de nivel, barreras vegetales, etc (figura 2.1).
- Los cambios de uso o cualidades que se produzcan entre dos espacios contiguos que tengan relevancia en el desarrollo del desalojo, como el cambio de una plaza a una avenida o el paso de un pasillo a un patio dentro de una edificación (figura 2.2).
- La diferencia entre un área segura y una peligrosa como el paso de una planta baja libre a una zona verde y despejada (figura 2.3).

Los límites permeables dentro de un `Space` son los que brindan a los agentes móviles la posibilidad de cambiar de espacio, es decir, un límite permeable puede ser atravesado.

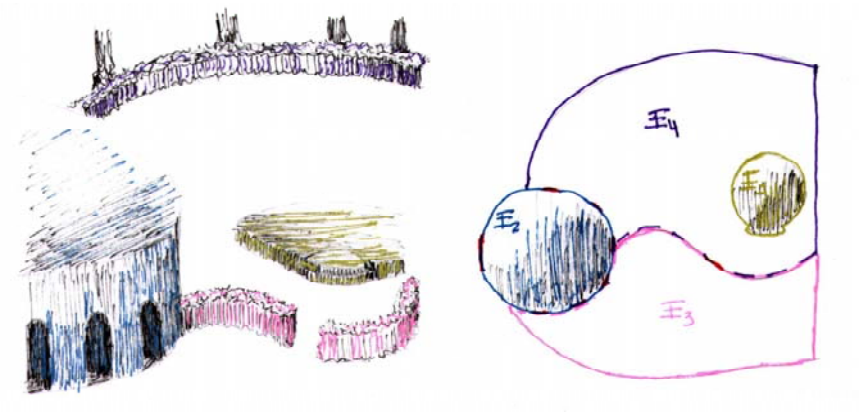

Figura 2.1: Objetos que definen los límites de un espacio al impedir el desplazamiento de los agentes móviles.

Este es el caso de puertas, áreas sin cerramientos, escaleras, ascensores, entre otros tipos de comunicación física entre dos espacios.

Una vez ubicados y definidos los límites de todos los espacios pertenecientes al modelo, el modelista deberá establecer para cada uno de los `Spaces` la manera como los agentes móviles se desplazarán dentro del espacio mediante la definición de la regla de movimiento (véase sección 4.1); también el modelista debe asignar el tamaño de cada unidad espacial o celda que conforma el espacio, el tiempo que tardarán los agentes móviles en atravesar cada uno de los límites permeables o `Doors`, el sentido en el que está permitido hacerlo y la cantidad de personas que podrán atravesarlas en el mismo instante de tiempo; entre otros parámetros.

Las figuras 2.4 muestran diferentes tipos de espacios y como se ha realizado el modelado en gSpaces de los límites permeables y no permeables. Cada uno de los espacios está envuelto en una línea de color que define el área que lo conforma.

En la figura 2.5 tenemos un ejemplo de como puede modelarse un sector urbano como un conjunto de espacios para la librería gSpaces.

Es importante destacar que para poseer información estadística de la cantidad de personas que se encuentran a salvo en un determinado instante de tiempo durante la simulación de un modelo, los cambios de uso determinan límites para los espacios y las zonas seguras deben ser definidas como espacios a parte con la finalidad de poder cuantificar la cantidad de individuos se han puesto a salvo en un instante de tiempo dado.

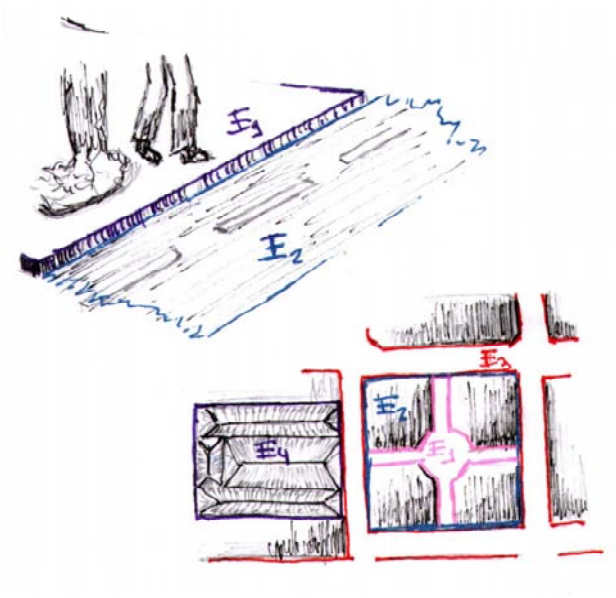

Figura 2.2: Límites definidos por los cambios de uso del espacio.

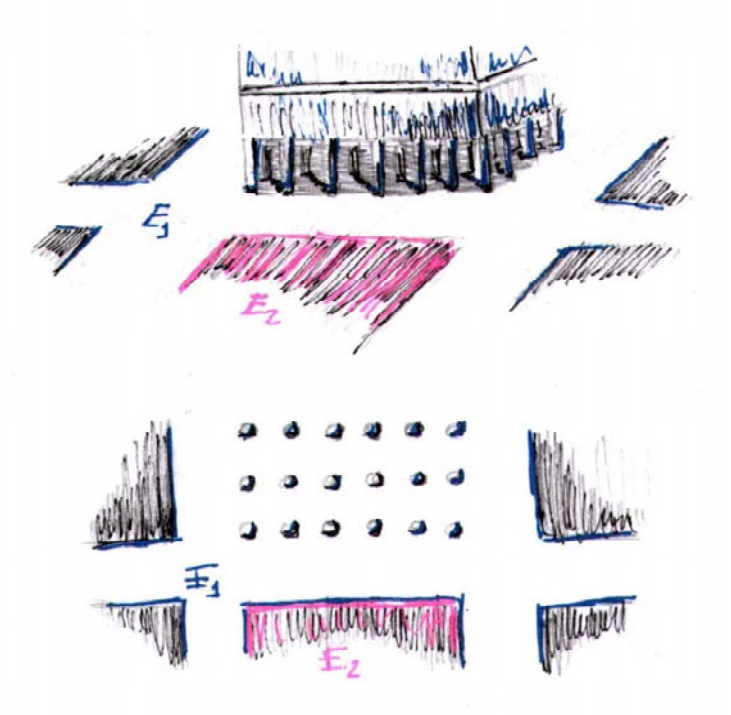

Figura 2.3: Límites de un espacio definidos por la seguridad que una determinada área puede brindar a los agentes móviles.

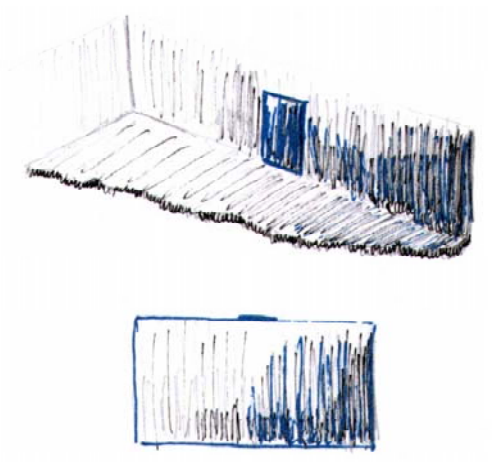

Figura 2.4: Espacio sencillo y los límites que lo definen. Un conjunto de límites no permeables `Walls` representados por la línea azul que lo envuelven y un límite permeable, `Door` que representa la salida del espacio.de sus límites.

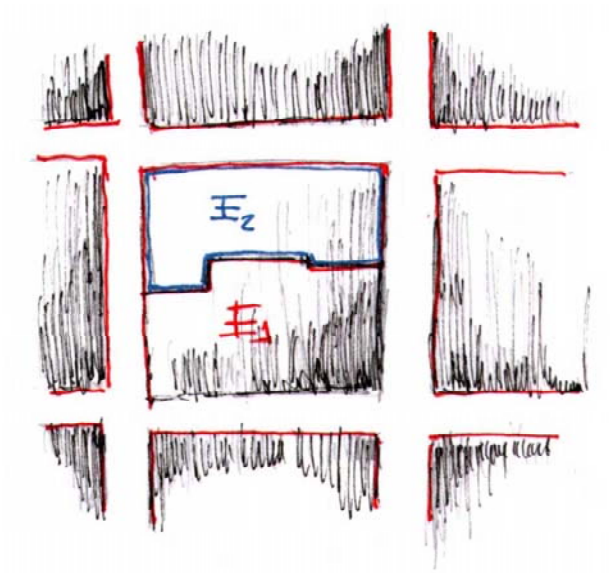

Figura 2.5: Sector urbano y las envolventes que delimitan los espacios. En la figura se observan 9 manzanas correspondientes a un sector urbano. Cada una de ellas fue modelada como un espacio a parte y la manzana central está dividida en dos espacios diferentes, el espacio E1 y el espacio E2 como se observa en la figura. Las calles pueden ser un espacio a parte, o varios espacios de acuerdo con lo que desee estudiar el modelista.

Odio a esta máquina, no hace lo que yo quiero sino lo que le digo que haga.
Lamento de un programador.

Capítulo 3

La librería gSpaces

En este capítulo se explica el contenido y la utilización de la librería gSpaces. Comenzaremos describiendo las características generales del meta-modelo y luego explicamos en detalle las partes que contiene un modelo realizado con nuestra librería. La utilización se de la librería se mostrará a través de códigos específicos para sus principales componentes. Se comienza mostrando el código básico de un espacio, luego mostraremos el código necesario para crear los individuos pertenecientes al modelo, los detalles para definir una regla de movimiento propia, cómo generar eventos que modifiquen el espacio en tiempo de ejecución, como crear la salida general del modelo y finalmente se muestran el código que contiene los métodos y variables propias de la ejecución del simulador GALATEA.

3.1. El Meta-Modelo gSpaces

Como se ha explicado anteriormente en la sección 1.4 del capítulo 1, gSpaces es una librería que permite desarrollar modelos que simulan, entre otras cosas, el desalojo de espacios arquitectónicos o urbanos. Puede decirse que es un Meta-Modelo porque se han modelado cualidades de los espacios y de los agentes móviles para que el modelo sea consistente, posteriormente el modelista se encarga de crear un modelo particular con las cualidades que desea estudiar. gSpaces posee un conjunto de objetos con características generales que brindan al modelista una librería que puede emplear para construir un modelo particular. La realización del modelo general o Meta-Modelo se basó en las siguientes planteamientos:

- No todas las personas ante una catástrofe natural son víctimas del pánico.
- Las personas o peatones avanzan con velocidades diferentes de acuerdo a sus capacidades y posibilidades.

- Cada peatón puede reaccionar de diferente manera al momento de desalojar un espacio.
- Los peatones pueden seguir el mismo camino por el que otros peatones se han desplazado anteriormente.
- El espacio puede ser modificado debido al cambio de su configuración formal o estructural a causa de una catástrofe.
- Un espacio posee cualidades generales que lo definen, ya sea arquitectónico o urbano.
- Un espacio puede ser modelado como un plano de dos dimensiones que contiene otros espacios.
- Las diferencias de nivel contenidas dentro de un mismo perímetro de paredes o límites pueden ser modeladas como diferentes espacios contiguos en caso de ser necesario.
- Los diferentes pisos de una edificación pueden ser modelados como diferentes espacios cuyos flujos de personas se comunican con los espacios que modelan los niveles inferior y superior.

En función de estas premisas, la librería gSpaces posee las siguientes características:

Espacio semicontinuo o híbrido. El espacio está dividido en *unidades espaciales* y es concebido como un plano donde las coordenadas son continuas dentro de los ejes de coordenadas x y y. De esta forma tanto la velocidad con la que se desplaza un individuo como su posición son continuas, pero la manera en la que el entorno influye en cada agente móvil depende de las unidades espaciales discretas en las que se dividió el espacio. Esta cualidad permite discretizar algunas de las variables y que otras continúen conservando continuidad, (véase sección 1.4).

Tiempo continuo con avance discreto. El tiempo es una variable continua que va siendo incrementada en instantes discretos. Las variables y los estados del sistema son evaluados y actualizados solamente en estos instantes de tiempo. Esta forma de modelar el tiempo se enmarca dentro del paradigma de simulación y modelado conocido como DEVS sobre el cual se basa GALATEA (véase sección 1.4). Este paradigma facilita la programación de las reglas de movimiento que se utilizan para determinar el nuevo estado del modelo.

Estado con variaciones discretas. Los eventos que se producen en el modelo son instantáneos. Esto hace que cada vez que un agente móvil actualiza su posición cambiando sus coordenadas x y y en forma instantánea y la posición de ningún agente cambiará mientras no haya un evento que provoque el cambio.

Determinístico o Probabilístico. Diferentes realizaciones de un mismo experimento con las mismas condiciones iniciales generan la misma salida si no se incluyen variables aleatorias en el modelo, pero en caso de tener alguna variable aleatoria el comportamiento del modelo será estocástico.

Individuos diferenciables con comportamientos particulares. En gSpaces cada uno de los agentes móviles es una entidad completamente diferenciable con características particulares y con la posibilidad de comportarse de una manera diferente con respecto a el resto de agentes móviles que se encuentran en la simulación. Los agentes móviles están basados en los mensajes de GALATEA donde cada individuo es un objeto diferente con valores particulares para sus características, que pueden variar durante la ejecución de la simulación. También se le pueden agregar nuevos atributos en caso de ser necesario. Debido a que son entidades diferenciables pueden comportarse de manera diferente ante un mismo estado del sistema si el modelista así lo contempla, e incluso existe la posibilidad de que se comporten como agentes inteligentes en el caso en que el modelista lo considere necesario.

Espacio modificable. Debido al tipo de catástrofes que se desea modelar por medio de gSpaces, es necesario que los espacios tengan la posibilidad de cambiar sus características y límites en tiempo de ejecución.

3.2. Código de un modelo gSpaces

Un modelo gSpaces está fundamentado en GALATEA que es la plataforma de simulación que soporta a la librería. GALATEA está basada en objetos y agentes; y "corresponde al estilo de modelado 'orientado a la red': los componentes del sistema modelado son organizados en redes de nodos que intercambian mensajes"[32].

3.2.1. Nodos que conforman un `Space` y su funcionamiento

Un espacio modelado con gSpaces es una subred que se inserta dentro de la red general de un modelo GALATEA. Con gSpaces cada espacio se modela mediante un `Space` dentro del cual hay dos nodos GALATEA mas un nodo por cada uno de los límites permeables que el espacio arquitectónico o urbano tenga. El primero de los nodos GALATEA es el nodo `Move`, heredero del nodo `Autonomous` de GALATEA, y encargado de actualizar las posiciones de los individuos pertenecientes a un espacio en cada iteración de tiempo de acuerdo a la regla de movimiento que el `Space` tenga. El nodo `Move` tiene una regla de movimiento *random walk* por omisión, pero el modelista puede programar o seleccionar otra regla de movimiento. El segundo nodo `Mobiles` se extiende del nodo `Gate` de GALATEA y actúa como contenedor de los agentes móviles que se encuentran dentro del espacio, es decir, contiene los mensajes que se encuentran dentro del espacio y sólo los envía a las salidas, es decir a los nodos `Doors`, si el nodo `Move` le solicita su envío a un nodo `Door` y el

nodo `Door` posee la capacidad suficiente para alojarlo. Cada una de las salidas del espacio se modela con un nodo `Door`, que es una extensión del nodo `Resource` de GALATEA, y se encarga de enviar a los individuos a nuevos espacios o a la salida general del modelo. Los nodos `Door`s se le pueden configurar para que retengan a los agentes móviles por un tiempo determinado por el modelista. Además, posee una capacidad máxima también definida por el modelista, y no le son enviados mensajes si su ocupación no es menor que su capacidad total.

La figura 3.1 muestra la definición de un espacio como un `Space` propio de la librería gSpaces. A la izquierda se muestra el espacio como es en el sistema real y a la derecha la abstracción realizada para los `Space`s de gSpaces.

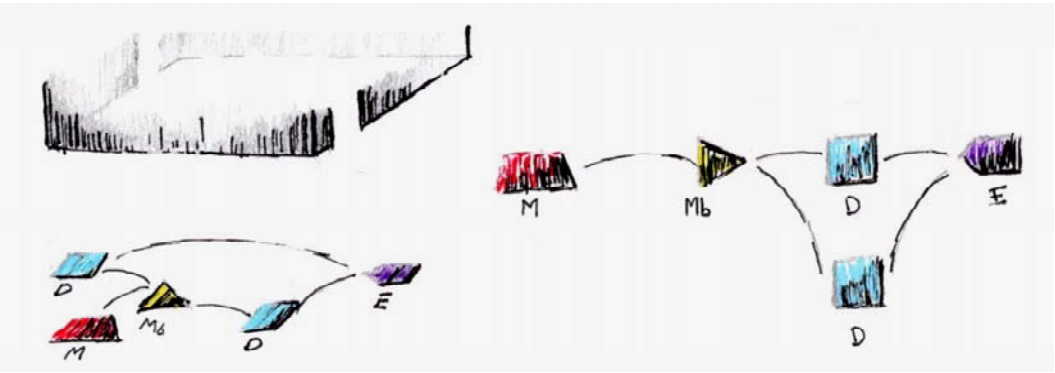

Figura 3.1: Espacio sencillo con dos salida llevado a nodos gSpaces. Cada espacio se transforma en un nodo `Space` que a su vez está compuesto por un nodo `Move`, un nodo `Mobiles`, tantos nodos `Door` como límites permeables tenga el espacio. También en el modelo hay un nodo `Exit` que permite retirar de la simulación a aquellos agentes móviles que han salido del sistema.

Un límite permeable, en lo que a términos del modelo se refiere, puede estar conformado por uno o dos nodos `Door`s. Será sólo un nodo `Door` si el límite comunica al espacio con la salida del modelo o es un límite que solamente puede atravesarse en una dirección. En cambio serán dos nodos `Door`s, si el límite permeable comunica dos espacios y puede ser cruzado en ambas direcciones. En este último caso cada uno de los nodos `Door`s pertenecerá a cada uno de los espacios comunicados. Esta dualidad permite mantener estadísticas diferentes para ambos sentidos de las puertas, bloquear uno de los sentidos y asignar índice de selección diferentes para cada uno de los sentidos.

Como ya se menciono anteriormente, todos lo nodos que modelan un espacio están contenidos en un envoltorio llamado `Space` que es el código al que tiene acceso el modelista siendo el nodo `Move`, los `Door`s y el `Mobiles` son atributos de `Space`. El funcionamiento de esta pequeña subred es como el de las colas virtuales de un banco en el que existe un conjunto de personas, representadas por los agentes móviles, que se encuentran esperando para ser atendidas por una de las taquillas, que en el caso de un `Space` son las puertas del espacio modeladas como nodos `Door`s. La máquina que se encarga de asignar las posiciones o los números correspondientes a la cola es modelada por el nodo `Move`.

El nodo `Move` actualiza las posiciones de los agentes móviles de acuerdo a la velocidad que obtiene de la regla de movimiento del `Space` e informa al nodo `mobiles` que individuos han atravesado algún límite permeable y pueden salir del espacio. Si existe capacidad disponible dentro del nodo `Door` correspondiente al límite interceptado por el agente móvil, el nodo `mobiles` ejecuta la método `sendto()` enviando al agente móvil al nodo `Door`.

3.2.2. Código necesario para modelar un espacio en gSpaces

El código necesario para modelar un espacio por medio del envoltorio `Space` está conformado básicamente por cuatro partes. Primero se invoca al constructor de `Space`, luego se le adicionan tanto los límites no permeables, mediante el método `addWall()`, como los límites permeables, con el método `addDoor()`, y por último se invoca al método `build()` que se encarga de construir el `Space`, creando una rejilla que cubre el espacio y determinando el resto de los atributos del `Space`.

A manera de ejemplo el código 1 se muestra la fracción de código correspondiente para modelar un espacio cuadrado de 20 unidades cuadradas de área y un puerta de 5

Code 1 Código necesario para modelar un espacio con la librería gSpaces.

```
import gSpaces.Space;
  ...
public class EjemploSimple {
  public static Space Cuarto = new Space("Cuarto", 55.0, 45.0, 2.0, 0.5);
  public static void main(String args[]){
    Cuarto.addWall(40.0, 40.0, 60.0, 40.0);
    Cuarto.addWall(60.0, 40.0, 60.0, 60.0);
    Cuarto.addWall(60.0, 60.0, 40.0, 60.0);
    Cuarto.addWall(40.0, 60.0, 40.0, 45.0);
    Cuarto.addDoor(40.0,40.0,40.0,45.0, exit,'U',4,0.0,1);
    Cuarto.build();
    ...
  }
}
```

unidades lineales, como se muestra en la figura 3.2. Lo primero que se debe hacer es crear un `Space` como atributo de la clase principal del modelo utilizando el constructor `Space()`, en este caso el nombre de atributo es `Cuarto`, como se observa en la tercera línea del código 1. El constructor `Space()` recibe como parámetros una cadena de caracteres que contiene el nombre del espacio, (`Çuarto"`); dos números reales que correspondientes a las coordenadas de un punto interno del espacio, (`55.0, 45.0`), el cual se utiliza para definir el área contenida por dentro del `Space`; un real que define la resolución o tamaño de sus celdas en el espacio, (`2.0`); y por último otro real que es el paso de simulación o tiempo entre activación asignado para el espacio en el modelo (`0.5`). Los criterios para seleccionar la resolución del espacio como el paso de simulación se estudian en la sección 3.2.3.

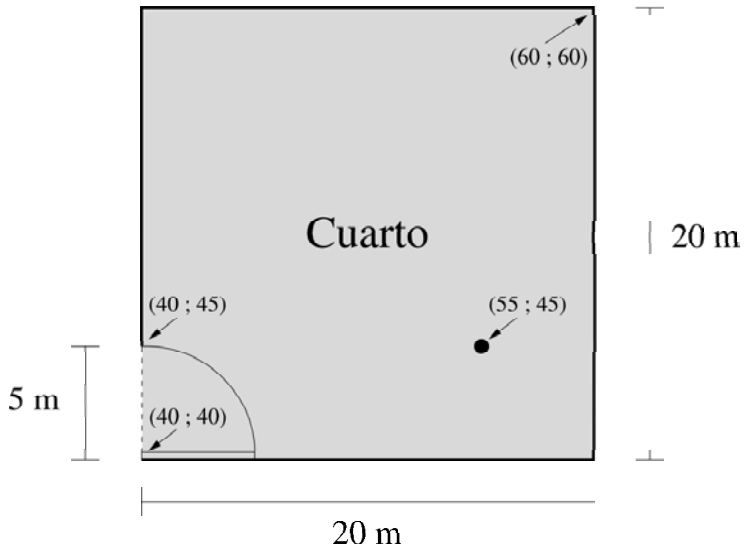

Figura 3.2: Ejemplo de un espacio simple con una única puerta. El punto se utiliza para determinar toda el área del espacio (gris). Las paredes, límites no permeables, están dibujados con una línea continua; mientras que la puerta, límite permeable, está dibujada con una línea segmentada.

Como los límites de los espacios pueden cambiar durante la simulación del modelo, estos se deben asignar dentro del método `main()` de la clase principal del modelo (línea 4 del código 1). Para agregar cada límite no permeable un espacio se utiliza el método `addWall()`, llamándolo por medio del `Space` donde se agregarán, en este caso se agregar 4 paredes a `Cuarto` invocando 4 veces el método `Cuarto.addWall()` (líneas 5-8 del código 1). En cada invocación el método recibe las coordenadas de los puntos que extremos que definen uno de los límites, por ejemplo, para el primer límite las coordenadas son (`40.0, 40.0, 60.0, 40.0`). El método le agrega a la lista de límites no permeables del objeto `Cuarto` una objeto `Wall` cada vez que se ejecuta.

Posteriormente se agregan los límites permeables del espacio, en este caso uno solo, mediante el método `Cuarto.addDoor()` en la línea 9 del código 1. El método recibe las coordenadas de los extremos del límite (`40.0, 40.0, 40.0, 45.0`); seguidos del nombre del nodo a donde se envían los agentes móviles que cruzan ese límite (`exit`), el tipo de de límite (`'U'`), su capacidad (`4`), la demora en cruzarlo (`0.0`) y el índice de selección de ese límite permeable (`1`). Hay tres tipos de límites permeables que pueden pertenecer a un espacio que serán explicados mas adelante en la sección 4.

Finalmente, en la línea 10 del código 1, se ejecuta el método `Cuarto.build()` que construye la rejilla de celdas del espacio `Cuarto` y calcula la distancia más corta que hay desde el centro de cada celda hasta cada uno de los `Walls` y `Doors` del espacio.

3.2.3. Paso de simulación y resolución de los Spaces

El tiempo en el simulador GALATEA está definido en unidades y cada modelista decide la equivalencia tendrán para su modelo. Esta relación siempre dependerá de las cualidades del modelo y cuanto tiempo transcurre dentro de sistema real para que se produzcan cambios. Por ejemplo en un modelo en el que cada año se pueden observar pequeños cambios en el estado del sistema, las unidades del simulador podrán ser el equivalente a años para que el tiempo de simulación se corresponda con los cambios de estado del sistema real. En el caso de desalojos de espacios el sistema a modelar es de tiempo continuo y la manera de simular esta cualidad con modelos de tiempo discreto es utilizar un paso de simulación lo suficientemente corto como para que no se dejen de registrar variaciones en el sistema. Una persona puede tomar en fracciones de segundo una decisión que varíe el estado del sistema y puede llegar a recorrer en un segundo una distancia comprendida entre ochenta centímetros y dos metros, por lo tanto es conveniente usar como unidades de simulación los segundos y actualizar el sistema en fracciones de segundo, por ejemplo 0.2 segundos como paso de simulación. Dentro de esta delicada relación que se refiere al tiempo que transcurre en el sistema para que se produzca un cambio de estado se ve involucrado del tamaño de las celdas. Las celdas poseen toda la información que en un instante determinado influencia a los agentes móviles que se encuentran dentro de ella; son el entorno inmediato de un individuo que se encuentra dentro de un espacio. Es necesario que el modelista contemple y planifique la relación entre el paso de simulación, la velocidad máxima de los individuos y el tamaño de las celdas con la finalidad del que el sistema no pierda realismo y cada individuo sea influenciado debidamente por su entorno inmediato de acuerdo con el sistema real. Por ejemplo, cuando una regla de movimiento no contempla que dos agentes móviles no pueden ocupar el mismo espacio, las celdas deberán poseer un área que pueda contener a dos agentes o más para que el modelo sea coherente con la regla de movimiento que se plantea.

El modelista, luego de determinar el paso de simulación que considera más adecuado para el sistema que desea modelar, debe asignar el paso de simulación dentro del constructor del `Space`.

3.2.4. Regla de movimiento por omisión

Los agentes móviles que se encuentran dentro de un espacio se desplazan a su nueva posición en función de una regla de movimiento que está basada el estado actual del agente y en las variables que puede brindar la celda como su entorno inmediato. Los `Spaces` poseen una regla de movimiento por omisión con la que los agentes móviles avanzan con una rapidez constante de una unidad de distancia por unidad de tiempo de simulación, cambiando de dirección de manera aleatoria. Los `Spaces` utilizan la regla de movimiento que tienen asignada por omisión si el usuario no asigna otra diferente. En caso de que el modelista desea asignar una nueva regla de movimiento para su modelo puede hacerlo de manera sencilla como se explica más adelante en la sección 4.1 de este capítulo.

3.3. El nodo `MobileAg` y los agentes móviles en gSpaces

En un simulador que modela por medio de redes de nodos y mensajes como GALATEA los agentes móviles son objetos que recorren la red de nodos pasando de unos a otros en función de reglas propias del sistema a modelar. Dichos objetos son creados dentro del modelo mediante un nodo. El nodo creador, llamado `MobileAg`, es un heredero del nodo `Input` de GALATEA y crea todos los agentes móviles que se encontrarán dentro del sistema. El modelo de simulación puede generan un evento que activa al nodo `mobileAg` en cualquier momento mediante el método `mobileAg.add()`. El código 2 muestra en la línea 3 como se declara el nodo creador `mobileAg` mediante el constructor de la clase `MobileAg`, pasándole el nombre del `mobileAg` como su único parámetro. También, se muestran cuatro maneras

Code 2 Declaración del nodo `mobilAg` y diversas formas de creación de agentes móviles.

```
import gSpaces.Space;
  ...
public class EjemploMobilAg {
  public static MobilAg mobilAg = new MobilAg("mobilAg");
  ...
  public static void main(String args[]){
    ... // Creacion de los espacios del modelo
    mobilAg.add(Pasillo, 2);            // Agrega 1 A.M. de tipo 2 en Pasillo
    mobilAg.add(Patio, 1, 20);          // Agrega 20 A.M. de tipo 1 en Patio
    mobilAg.add(Salon, 3, 30.0, 40,0); // Agrega 1 A.M. de tipo 3 en Salon
                                        // en la posicion (30;40)
    mobilAg.add(100, Pasillo, 1, 10);   // Agrega 10 A.M. de tipo 1 a Pasillo
                                        // cuando el tiempo de sim. sea 100
    ...
  }
}
```

diferentes de llamar al método de creación de agentes móviles. La primera crea un agente móvil de tipo 2 ubicado de forma aleatoria dentro del `Space Pasillo`; la segunda crea 20 agentes móviles de tipo 1 en `Patio`, ubicándolos también aleatoriamente en ese espacio; la tercera crea un agente de tipo 3 en la posición (30,0;40,0) del `Salon`; y por último, la cuarta es igual a la primera con la diferencia de que la creación no ocurre inmediatamente sino cuando el tiempo de simulación llegue a 100 unidades.

Los agentes móviles de gSpaces son mensajes GALATEA con algunos campos para que puedan ser manipulados por la librería como lo son su posición del agente, su velocidad, su tipo, la visibilidad y, en caso de existir, el límite permeable que atravezaría si avanzara según se lo indica su velocidad. Sin embargo, es posible que el modelista agregue nuevos campos a los agentes móviles de gSpaces sobre-escribiendo el método `creating` en una nueva clase heredera de `MobileAg`. El nuevo método deberá ejecutar el método `creating`

perteneciente a su clase superior y agregar nuevos campos a los individuos utilizando la instrucción `addField` de GALATEA. Realizar este proceso puede ser necesario cuando se programan reglas de movimiento particulares, diferentes a la que la librería trae.

3.4. Nodos `Doors` y espacios múltiples

Un espacio compuesto por varios sub espacios también puede ser modelado por la librería, como se muestra en la figura 3.3.

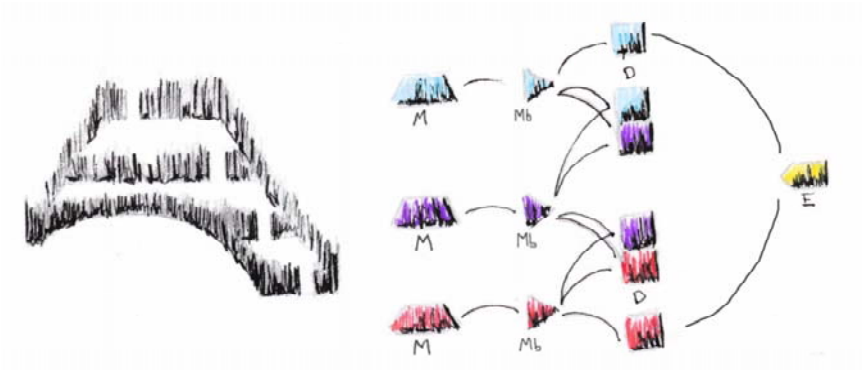

Figura 3.3: Espacio compuesto con una salida llevado a nodos gSpaces. Como puede observarse en la figura, cada uno de los espacios posee dos puestas que son modeladas como nodos `Door`, un nodo `Move` y un nodo `Mobiles`. Cada uno de los espacios está representado con un color, azul, morado y rojo según la ubicación que se observa a la izquierda de la figura. Las puertas son compartidas por dos espacios diferentes contiguos, porque al comunicarlos, pertenecen a ambos y son clasificadas como bidireccionales, véase 5.2.3).

Como se explica en la sección 2.2 un modelo gSpaces está basado en abstracciones de espacios arquitectónicos o urbanos que, después de pasar por el proceso de modelado, generan una combinación de espacios propios de nuestra librería que se comunican unos con otros a través de límites permeables llamados `Doors`. El nodo `Door` define un límite permeable para un espacio, entendiéndose como permeabilidad a la la capacidad que poseen el nodo de enviar mensajes provenientes de un espacio hacia otro espacio contiguo, es decir, estos límites pueden ser atravesados por los agentes móviles. Esta cualidad los convierte en la comunicación entre dos espacios modelados con nuestra librería. Los nodos `Door` están definidos espacialmente por las coordenadas de los extremos de un segmento de recta y son agregados a un `Space` mediante el método `addDoor()`. Para realizar el modelado de la comunicación entre espacios los nodos `Doors` pueden ser creados de maneras diferentes de acuerdo a sus cualidades dentro del sistema real definiendo de esta forma los tipos de puertas que existen en g`Space`.

El primer tipo son las **puertas unidireccionales** que sólo pueden ser recorridas por los agentes móviles en un sentido no permitiendo la circulación en el sentido contrario. Pueden comunicar dos espacios contiguos o dirigir los mensajes a la salida general del modelo (véase

3.7). Una puerta unidireccional se agrega a un `Space` utilizando el método `addDoor` como lo muestra el código 3.

Code 3 Agrega una puerta unidirecional al espacio `Cuarto` que envía a los agente a `exit`

```
Cuarto.addDoor(40.0,40.0,  // Pos. del primer extremo de la puerta
               40.0,45.0,  // Pos. del segundo extremo de la puerta
               exit,       // Nodo destino de los agentes moviles
               'U',        // Tipo de puerta (U: unidirecional)
               2,          // Capacidad maxima 2 agentes
               0.1,        // Retardo de 0.1 unidades de tiempo
               1.0)};      // Indice de seleccion
```

El método agrega un `Door` al `Space Cuarto`, recibe como parámetros los dos pares coordenadas que definen la puerta como un segmento de recta, `(40.0,40.0)(40.0,45.0)`; el espacio o nodo hacia el que se dirigen los agentes móviles, `exit`; el tipo de `Door`, en este caso unidireccional o tipo `'U'`; la capacidad del `Door` medida en la cantidad de agentes móviles que pueden atravesarla en un mismo instante de tiempo, `2`; el tiempo que tardan en hacerlo, `0.1`; y el índice de selección, `1.0`.

El segundo tipo de `Doors` son las **puertas bidireccionales**, que comunican a dos espacios en ambos sentidos. Son también creadas y agregadas a un `Space` por método `addDoor` pero con un parámetro más que indica el índice de que la puerta sea seleccionada por los agentes móviles ubicados en el nodo destino para ir al nodo origen, como se ve en el código 4. Las puertas bidireccionales son modeladas mediante dos nodos `Doors`, uno en

Code 4 Agrega una puerta bidirecional a `Cuarto` que lo comunica con `Sala`

```
Cuarto.addDoor(40.0,40.0,  // Pos. del primer extremo de la puerta
               40.0,45.0,  // Pos. del segundo extremo de la puerta
               Sala,       // Nodo destino de los agentes moviles
               'B',        // Tipo de puerta (U: unidirecional)
               2,          // Capacidad maxima 2 agentes
               0.1,        // Retardo de 0.1 unidades de tiempo
               1.0,        // Indice de seleccion Cuarto -> Sala
               0.5)};      // Indice de seleccion Sala -> Cuarto
```

cada uno de los espacios que se estén comunicando. Los dos nodos `Doors` comparten su capacidad máxima y a los agentes móviles que están cruzándolos, es decir, que si un agente móvil está cruzando la puerta desde el `Cuarto` a la `Sala` éste ocupará tanto a la puerta del `Stace Cuarto` como a la del `Space Sala`. Los nodos `Door` bidireccionales arrojan para cada una de las direcciones estadísticas diferentes propias de su nodo padre `Resource` de GALATEA, debido a que al agregar una puerta bidireccional a un espacio se crean dos nodos, uno para cada uno de los espacios comunicados y cada uno de estos nodos posee estadísticas propias.

Es importante destacar que los índices de selección deben ser cuidadosamente manejados cuando en la regla de movimiento se quiere establecer una vía de desalojo que no sea a través de la puerta más cercana. La idea del índice de selección de una puerta nace de que en algunos espacios arquitectónicos es probable que exista un *r*ecorrido de desalojo que ha sido planificado en el momento de diseño y que sólo algunos tipos particulares de agentes móviles utilicen el sentido contrario a esta disposición, como por ejemplo los bomberos. Los recorridos de desalojo pueden definirse con puertas unidireccionales o con puertas bidireccionales que poseen probabilidad cero en uno de sus sentidos.

3.5. Cambios en la configuración espacial de los Spaces

La librería gSpaces ofrece la posibilidad de causar cambios de configuración formal a los espacios y de comportamiento a los agentes móviles. Debido a esta capacidad de cambiar el estado del sistema en tiempo de ejecución ha sido llamado `Seismic`. Dicho nodo debe ser creado dentro de modelo de la misma manera que los otros nodos propios de la librería, por medio de su constructor.

En gSpaces es posible simular los posibles cambios que puede sufrir un espacio, por ejemplo a raíz de un evento sísmico y los efectos que estos cambios tiene sobre la dinámica de los agentes móviles que están dentro de dicho espacio. A modo de ejemplo, en esta sección se creó el nodo `Seismic`. Dicho nodo es un heredero del nodo `autonomous` de GALATEA; posee un conjunto de métodos que pueden ser ejecutados durante la simulación y que permiten modificar los espacios pertenecientes al modelo y cambiar la velocidad de los individuos contenidos en ellos. El nodo `Seismic` fue creado como un ejemplo para mostrar como se pueden realizar modificaciones a los espacios en tiempo de ejecución, sin embargo, el modelista puede programar los cambios que desee utilizando un nodo `autonomous` de GALATEA que modifique las cualidades de los espacios pertenecientes al modelo generando eventos dentro de la dinámica del simulador.

Para utilizar al nodo `Seismic` luego de crearlo, se debe crear un arreglo de `doubles` que contenga los tiempos en los que van a ocurrir cada uno de los eventos que modifican el espacio. También, se debe modificar el método `fact` del nodo agregando los métodos que se deseen ejecutar colocando para cada uno el tiempo en que deben ser ejecutados, posteriormente se programa la activación del nodo en la clase principal generando el primer evento y posteriormente el nodo se encarga de ejecutar cada método en el tiempo correspondiente. A modo de ejemplo del método `fact`, el código 5 muestra las acciones realizadas por el nodo `Seismic` que se ejecutan en los tiempos 2, 3 y 4. El primer método ejecutado simula la caída de una de las paredes del `Espacio1` del modelo `Gsample1` en el tiempo 2, luego en el tiempo 3 un espacio es bloqueado por escombros y queda aislado del resto, esto se simula creando un nuevo límite no permeable para el espacio y finalmente en el tiempo 4 el 25 por ciento de las personas que se encuentran dentro del espacio resultan heridas y disminuyen su velocidad promedio a 0.5 metros por segundo.

Code 5 Función `fact` del nodo `Seismic`.

```
public boolean fact(){
  removeWall(Gsample1.w,Gsample1.Espacio1,Gsample1.Espacio2,10,1,0,2);
  createWall(20,20,80,80,Gsample1.Espacio1,3);
  peopleHurt(Gsample1.Espacio1,0.25,0.5,4);
  super.fact();
}
```

Cada uno de los métodos citado anteriormente se encuentran explicados en el archivo Seismic.html adjunto a la librería gSpaces junto con algunos otros métodos de la clase `Seismic` que pueden ser usados por el modelista.

3.6. El nodo `display`

Es el nodo encargado de dibujar gráficamente la dinámica ocurrida en un modelo gSpaces. El nodo `display` es heredero del *Autonomous* de GALATEA y su función es mostrar las actualizaciones del modelo de manera gráfica para cada paso de simulación o para cada intervalo de tiempo que el modelista desee. El código para incluir el nodo display en el modelo consta de su constructor y la sentencia para agregarle los espacios que van a ser mostrados gráficamente por medio de un applet java. Básicamente el código necesario para incluirlo en un modelo realizado con nuestra librería se muestra en el código 6.

Code 6 Activación de la animación.

```
...
public static Display display = new Display("Display",0.2);
/**Spaces to be displayed*/
display.addSpace(AreaSegura);
display.addSpace(PasilloE2);
display.addSpace(PatioE2);
display.addSpace(Edificio1);
...
```

El constructor recibe como parámetros el nombre del nodo y el intervalo de tiempo en el que se refrescará la ventana de animación. Posteriormente le son agregados los espacios que serán dibujados por el nodo. La manera como ha sido programado permite mostrar todos los espacios que pertenecen al modelo o sólo algunos de ellos de acuerdo a lo que el modelista desee.

3.7. El nodo `Exit`

Es el encargado de eliminar los mensajes que dejan de estar dentro del sistema al salir de los espacios que lo conforman hacia alguna área externa a las representadas por el modelo.

Se trata del nodo tipo `Exit` propio de GALATEA. Aquellos modelos desarrollados con gSpaces en los que los mensajes llegan a salir del sistema deben poseer un nodo `Exit` como parte de la red. Nos referimos a un nodo `Exit` como la salida general del modelo que se crea con un constructor que recibe el nombre del nodo y una letra que le indica al simulador el tipo de nodo que debe crearse de la forma siguiente:

```
public static Node exit = new Node(.Exit",'E');
```

Puede decirse que un mensaje será enviado a la salida general del modelo cuando su ubicación dentro del modelo deja de poseer relevancia y se considera que debe salir del sistema, por ejemplo, cuando un agente móvil ha desalojado la edificación en estudio y se encuentra lo suficientemente lejos del peligro como para considerarse a salvo.

3.8. Código propio del simulador y su ejecución

Una vez que han sido creados todos los nodos pertenecientes al modelo se debe colocar el código que genera las activaciones de los nodos pertenecientes a la red que simula el sistema y se tiene que además asignar las variables propias del simulador. Por ejemplo en el código 7 tenemos la asignación del título del modelo y el tiempo de simulación; la

Code 7 Código estándar de GALATEA para un modelo con gSpaces

```
class Gsample1{
  ...
  public static void main(String[] args){
    ...
    Glider.setTitle("Gsample1");      // Titulo del modelo
    Glider.setTsim(25);               // Tiempo de la simulacion
    GRnd.inisem();                    // Inicializacion de la semilla
    Glider.trace("Gsample1.trc");     // Achivo que almacena la traza
    Glider.stat("Gsample1.sta");      // Achivo que almacena las estadisticas
    Glider.act(creator,0);            // Activacion del nodo creator
    Glider.act(Espacio2.getMove(),1); // Primera activacion del Espacio2
    Glider.act(Espacio1.getMove(),1); // Primera activacion del Espacio1
    Glider.act(display,0);            // Primera activacion del display
    Glider.act(sismo,1);              // Primera activacion del sismo
    Glider.process();                 // Procesamiento de la red de nodos
  }
}
```

inicialización de la semilla del generador de número aleatorios; y luego está la asignación de los nombres de los archivos que almacenarán la traza y las estadísticas; después está la activación de cada uno de los nodos que lo requieren y finalmente la llamada al método propio de la clase Glider para el procesamiento de la red de nodos.

Para que el simulador pueda ser ejecutado el archivo galatea.jar debe estar incluido en la variable de entorno `CLASSPATH`. El simulador será activado cada vez que se ejecute en una

máquina virtual java la clases de `galatea` o `gSpaces` y contenga las instrucciones propias del simulador. En cada uno de los modelos el simulador es activado en la clase principal del modelo por el método `main` con la instrucción `Glider.process()`. Para compilar la clase correspondiente al modelo en una consola o shell correspondiente al sistema operativo se utiliza el comando:

`javac` *modelo*`.java`

donde `javac` es el compilador java y *modelo*`.java` es el nombre del archivo donde se programó el modelo. El comando hay que ejecutarlo desde el directorio donde se encuentra *modelo*`.java`. Una vez que con el comando anterior se ha creado la clase principal del modelo *m*odelo`.class` se corre el modelo mediante la instrucción

`java` *modelo*

con lo que se activará la ventana de animación propia de la librería y se arrojarán los resultados en los archivos *modelo*`.sta` y *modelo*`.trc` dentro del mismo directorio donde se encuentra la clase principal del modelo.

Eppur si muove
Galileo Galilei

Capítulo 4

Modelando el comportamiento de los agentes móviles

En este capítulo se explica como un modelista puede definir el comportamiento de los agentes móviles al modelar con gSpaces, como se define una regla de movimiento particular para un espacio o un modelo y se muestran varios ejemplos que están incluidos junto a los códigos que acompañan nuestra librería.

Como se había mencionado anteriormente en la sección 3.2.1 el nodo `move` perteneciente a los `Space`s de un modelo de nuestra librería se encarga de actualizar las posiciones para cada uno de los individuos que se encuentran dentro del espacio. Dicha actualización la realiza de acuerdo a una regla que propone una nueva velocidad para cada agente móvil, que la librería utiliza junto con el paso de simulación del `Space` se utiliza para realizar el desplazamiento del agente, asignándole a éste su nueva posición, siempre y cuando el agente al moverse no intercepte ninguno de los límites del `Space` en el que se encuentra. En el caso de que el eventual desplazamiento del agente intercepte alguno de los límites no permeables del `Space` el nodo `move` no moverá al agente y le asignará en el atributo `collision` el objeto de tipo `Wall` con el que chocaría el agente si se moviera. Pero, si la intersección es con un límite permeable y el nodo `Door` asociado al límite tiene capacidad suficiente para recibir al mensaje, el nodo `move` moverá al agente hasta el límite y le indicará al nodo `mobiles` que el agente saldrá del espacio a través de ese nodo `Door`. Sin embargo, si la capacidad del nodo `Door` es insuficiente para recibir al agente, el nodo `move` actuará como si el agente hubiese interceptado a un límite no permeable.

4.1. Definición de reglas de movimiento propias

Los espacios arquitectónicos y urbanos pueden poseer una dinámica propia que los diferencian y hacen que el comportamiento de los individuos que los habitan esté estrechamente relacionado con el espacio. Las personas no tienen el mismo comportamiento en un pasillo que en una plaza, y tampoco en una sala de exposiciones y un salón de clases, aunque alguno de estos espacios posean las mismas características formales. Debido a esta cualidad de los espacios, la librería gSpaces permite asignar una regla de movimiento diferente para cada uno de los espacios pertenecientes al modelo cuando el modelista lo considere necesario.

Si el modelista desea crear una regla particular de movimiento, en cambio de utilizar la regla que por omisión traen los `Space`s, debe tener en cuenta que el nodo `move` no permitirá que ningún agente móvil cruce ningún límite permeable que no tenga capacidad para recibir al agente ni tampoco cruce ningún límite no permeable. Cada `Space` puede tener una regla de movimiento diferente, la cual se asocia al `Space` en el momento de su creación, como se muestra en el código 8. Note que el constructor del `Space` tiene un parámetro más en comparación con el constructor que mostramos en el código 1. Este parámetro adicional, `SiChocoGiro`, es el nombre de la regla de movimiento que actualizará las velocidades de los agentes que se encuentren en el `Space Cuarto`.

Code 8 Constructor de un `Space` asociando a una regla de movimiento.

```
public static Space Cuarto =
               new Space("Cuarto",       // Nombre del espacio
                         50.0, 45.0,     // Pnto. interno del espacio
                         1.0,            // Resolucion espacial
                         1.0,            // Paso de simulacion
                         "SiChocoGiro"); // Regla de movimiento
```

El código de la regla de movimiento debe estar escrito dentro de una clase Java cuyo nombre tiene que ser el mismo que el dado a la regla de movimiento en el constructor del `Space`. La clase debe tener dos métodos, el constructor, que no recibe ningún parámetro; y un método llamado `move`, que contiene la regla de movimiento de los agentes. A manera de ejemplo, el código 9 muestra la clase java `SiChocoGiro` cuyo método `move` básicamente lo que hace es modificar la velocidad con la que venía el agente móvil si éste, al intentar moverse en el instante anterior, colisionó con un límite no permeable.

En general, las reglas de movimiento pueden definirse de acuerdo al estado del sistema en el momento de la actualización. Para esto las celdas y los agentes móviles poseen diversas variables que el modelista puede tomar en cuenta para seleccionar la nueva velocidad. Las celdas brindan por medio de sus atributos información sobre el entorno como:

- Cantidad de agentes móviles que contiene la celda.

Code 9 Clase `SiChocoGiro`, ejemplo de regla de movimiento definida por el modelista.

```
/*
 * Created on 08/02/2005
 */
import galatea.gSpaces.*;
import galatea.glider.*;
/**
 * @author Klaudia Laffaille
 *
 * Clase donde se define una regla de movimiento para el desplazamiento
 * de los agentes moviles dentro del espacio. Cada agente se mueve a
 * velocidad constante hasta que alcanzan un Wall, entonces gira 90 grados.
 */
public class SiChocoGiro{
  /* Constructor */
  public SiChocoGiro(){}
  /**Move -
  * Metodo que actualiza las posicion del agente movil desplazandolo en
  * a velovidad constante en linea recta hasta que alcance un limite no
  * que provoca un giro en su direccion de 90 grados a la izquierda.
  * @param e - Nodo move del espacio dentro del que se encuentra el mensaje.
  * @param m - Mensaje que cambiara su velocidad.
  * @param c - Celda en la que se encuentra el mensaje.
  * @return vel - Arreglo que contiene el nuevo vector velocidad del mensaje
  */
  public double [] move(Move e,Message m, Cell c){
    /* Arreglo de 2 posiciones que contiene el vector velocidad
     */
     vel [] nCoord = new double[2];
    // Si con la velocidad que venia el mensaje se alcanzo un Wall se gira
    // 90 grados en el sentido de las agujas del reloj
    if(m.getValue("Collision") instanceof Wall ){
      vel[0] = -m.vy;   // vx = -vy
      vel[1] =  m.vx;   // vy =  vx
    } else {            // Sino se sigue con la misma velocidad
      vel[0] =  m.vx;   // vx = vx
      vel[1] =  m.vy;   // vy = vy
    }
    return vel;
  }
}
```

- La distancia más corta a cada una de las `Walls` pertenecientes al espacio que contiene a la celda y la referencia al objeto en sí.
- Las `Doors` ordenadas por su índice de selección y la distancia desde el centro de la celda al punto más cercano a ella.
- La dirección hacia la que se dirigirán con mayor probabilidad los agentes móviles de la celda.
- La dirección que siguió el último agente que se movió dentro de la celda.
- El índice de obstáculos de la celda.

Los agentes móviles también poseen un conjunto de campos que pueden ser utilizados por el modelista para establecer diferencias o particularidades en la regla de movimiento que fueron citados en la sección 3.3. Con la posibilidad de utilizar todas estas variables, el modelista puede programar una o varias reglas de movimiento en lenguaje `java` que propongan una nueva velocidad para los agentes móviles pertenecientes al modelo.

El método `move()` recibe como parámetros tres objetos , el primero es el nodo `move` del `Space` asociado a la regla; el segundo es el agente móvil, que es un `Message` GALATEA; y el tercero es la `Cell` donde se encuentra el agente. El método calcula y devuelve un arreglo de `doubles` con dos elementos que corresponden a la nueva velocidad del agente móvil.

El nodo `move` del `Space` utilizará esta velocidad para intentar mover al agente en línea recta durante el tiempo que dure el paso de simulación del `Space` y si el movimiento es exitoso, el nodo `move` actualizará la posición del agente.

El modelista puede saber si en el intento de movimiento anterior el agente colisionó o intersecto algún límite del `Space` mediante el método `m.getValue(çolision")` donde, `m` es el mensaje que se está procesando. Este método devuelve el objeto que fue interceptado durante el recorrido del agente móvil. En el código 9 este método es utilizado para saber si el límite interceptado es de la clase `Wall`.

Existe un método de `Space`, llamado `checking()`, que se utiliza para chequear si con la nueva velocidad el agente móvil colisionará o no con alguno de los límites del `Space`. El método recibe como parámetros el agente móvil, la celda donde se encuentra el agente y un arreglo de dos `doubles` con la velocidad propuesta; y devuelve un doble con el tiempo que debe transcurrir para que una colisión ocurra.

El código 10 muestra la segunda regla que utilizaremos como ejemplo, `NearestDoor`, que consiste en un movimiento que dirige a los agentes hacia el límite permeable cuyo índice de ser seleccionada sea mayor y que además se encuentre más cerca de la posición actual del agente. Dicho proceso lo realiza valiéndose de uno de los atributos de `Cell`, la lista de `Doors` pertenecientes al `Space`, las cuales están ordenadas de menor a mayor distancia al centro de la `Cell` y en segundo lugar de acuerdo al índice de selección. Una vez obtenida la puerta más cercana los mensajes utilizando el método `c.getDoor(0)`, se calcula el punto medio de la puerta a partir de sus extremos y con este punto se calcula la distancia del agente móvil a la puerta. Para darle más realismo al movimiento, se conserva el 10% de

Code 10 NearestDoor, regla de movimiento que dirige a los agentes móviles a la puerta mas cercana y con la mayor índice de selección asignada.

```
/*
 * @(#)CloserDoor.java Created on 03/07/2005  Klaudia Laffaille
 *
 * Copyright (c) Galatea Group. Universidad de Los Andes.
 *
 */
import galatea.gspaces.*;
import galatea.glider.*;
/**
 * @author Klaudia Laffaille
 *
 * Clase que define una regla de movimiento que dirige a los agentes moviles
 * hacia la Door mas cercana al centro de la Cell donde se encuentren.
 */
public class NearestDoor {
  /**Constructor por defecto de la clase*/
  public NearestDoor(){}
  /**Mover -
    * Metodo que actualiza la velocidad de los agentes moviles en cada
    * iteracion de tiempo. En la regla planteada, los agentes moviles se
    * dirigen a la puerta mas cercana manteniendo su rapidez constante y
    * un 10% de su velocidad previa para darle mas realismo al movimiento.
    * @param m - mensaje al que se le actualizaran las posiciones.
    * @param c - Cell donde se encuentra el agente movil.
    * @return nCoord - arreglo de cuatro posiciones que almacena:
    * vel[] - nueva velocidad del mensaje.
    */
  public double [] move(Move e,Message m, Cell c){
    double [] vel = new double[2];
    double speed = Math.sqrt(m.vx*m.vx + m.vy+m.vy); // rapidez del agente
    Door nearestDoor = c.getDoor(0);                  // puerta mas cercana
    double [] doorCenter = new double[2];             // centro de la puerta
    doorCenter[0] = (nearestDoor.x1+nearestDoor.x2)/2.0;
    doorCenter[1] = (nearestDoor.y1+nearestDoor.y2)/2.0;
    double [] d =  new double[2];                    // distancia a la puerta
    d[0] = m.x - doorCenter[0];
    d[1] = m.y - doorCenter[1];
    dModule = Math.sqrt(d[0]*d[0] + d[1]*d[1]);      // Modulo de la distancia
    vel[0] = 0.9*d[0]*speed/dModule + 0.1*m.vx;      // Velocidad en x
    vel[1] = 0.9*d[1]*speed/dModule + 0.1*m.vy;      // Velocidad en x
    return vel;
  }
}
```

la velocidad previa siempre manteniendo la rapidez constante. En la figuras 4.1 se muestra dos estados de la simulación del modelo utilizando la regla del código 10. Puede observarse como los agentes se dirigen a la `Door` que posee el `Space`.

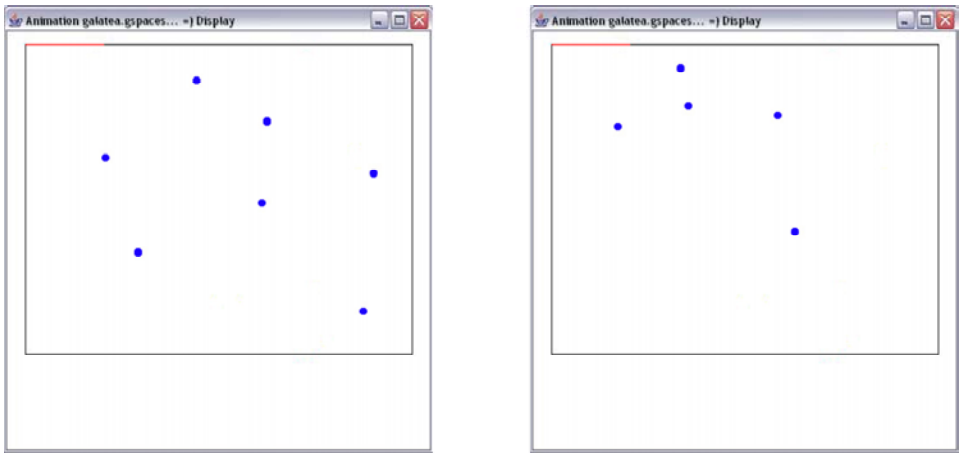

Figura 4.1: Dos vistas de instantes de tiempo diferentes de la animación. El tiempo avanza de izquierda a derecha.

Como tercer caso explicaremos como se puede establecer una regla de movimiento diferente para cada tipo de agente. Como primer paso se deben crear varios tipos de agentes móviles como se mostró en el código 2. Para nuestro ejemplo crearemos tres tipos de agentes móviles. El método `move()` de la clase java llamada `DiferentMobileAgents`, que se muestra en el código 12, contiene una regla de movimiento que asignará diferentes reglas de movimiento de acuerdo al tipo de agente que esté actualizando su velocidad. El tipo de agente se obtiene mediante el método `m.getintValue("tU").` Si el agente es de tipo `0` su velocidad será calculada utilizando el método `move` que tiene `Space` por omisión. En cambio, si es de tipo `1` la velocidad sera calculada siguiendo la regla de la clase `NearestDoor`. Por último, si el agente es de tipo `2` se quedará paralizado.

En las figuras 4.2 puede observarse el comportamiento de los agentes en función del tipo. Los agentes de tipo `0`, color azul, se mueven con direcciones aleatorias, los de tipo `1`, color verde, se dirigen hacia la puerta y los de tipo `2`, color rojo, conservan su posición inicial.

El cuarto caso que estudiaremos es la posibilidad de hacer cambiar el comportamiento de los agentes móviles en un determinado instante de tiempo. Este caso es muy útil en la simulación de desalojo luego de un evento catastrófico. El método `move()` de la clase java llamada `runOut`, que se muestra en el código 13, contiene una regla de movimiento que asigna la velocidad de forma diferente de acuerdo al instante de tiempo en el que esté la simulación. El tiempo de simulación se obtiene mediante el método `Glider.getTime()` y si este tiempo es menos que 20 unidades los agentes se moverán aleatoriamente usando la regla de movimiento por omisión que tiene el `Space`. Pero, cuando el tiempo sobrepasa las 20 unidades el comportamiento de los agentes cambia, dirigiéndose a la puerta más cercana, según lo indique el método `NearestDoor.move().`

Code 11 Método `move` de la clase `DiferentMobileAgents`.

```
/*@(#) DiferentsMovilAgents  Created on 03/07/2005  Klaudia Laffaille
 * Copyright (c) Galatea Group. Universidad de Los Andes.
 */
package demos.desalojos;
import galatea.gspaces.*;
import galatea.glider.*;
/**
 * @author Klaudia Laffaille
 *
 * Clase que define una regla de movimiento que asigna comportamientos
 * diferentes para cada tipo de agente movil que posee el modelo. Los
 * agentes tipo 0 se moveran con la regla de movimiento por omision que
 * posee la libreria (RamdomWalk), los tipo 1 se dirigiran a la puerta
 * con mayor probablidad de seleccion asignada que se encuentre mas cerca
 * y los tipo 2 se quedaran en su posicion inicial victimas del panico.
 */
public class DiferentsMovilAgents {
/**Constructor por defecto de la clase*/
public DiferentsMovilAgents(){
}
public double [] Move(Move e,Message m, Cell c){
double [] nCoord = new double[4];
int agentTipe = m.getintValue("tU");
switch(agentTipe){
case 0:
nCoord = c.getMove().move(m,c);
break;
case 1:
nCoord = this.CloserDoor(e,m,c);
break;
case 2:
nCoord = this.Freeze(e,m,c);
break;
}
return nCoord;
}
public double [] CloserDoor(Move e,Message m, Cell c){
     double [] nCoord = new double[4];
     double df = -Double.MAX_VALUE;
double cosA = -Double.MAX_VALUE;
double senA = -Double.MAX_VALUE;
double x = -Double.MAX_VALUE;
double y = -Double.MAX_VALUE;
     int vis = m.getintValue("vi");
int ind = 0;
Dist [] d =  (Dist[]) c.getDDoors();
Dist di = d[ind];
Door p = (Door) di.getOb();
double dp = di.getDist();
double []pointD = di.getPinObj();
boolean close = false;
ind++;
```

Code 12 Continuación del código

```
if (((p.getClose() == true)&&(vis>dp)&&(ind<c.getDDoors().length))||
((close == true)&&(ind<c.getDDoors().length))){
while(close==true){
di = d[ind];
p = (Door) di.getOb();
dp = di.getDist();
pointD = di.getPinObj();
close = p.getClose();
if((vis>dp)&&(p.getClose()==true)){
ind++;
}else{
close = false;
}
}
}
x = pointD[0];
y = pointD[1];
double dr = m.getdoubleValue("v")*e.getIt();
double dx = 0.9*(x-m.getdoubleValue("x"))+0.1*m.getdoubleValue("vx")*e.getIt();
double dy = 0.9*(y-m.getdoubleValue("y"))+0.1*m.getdoubleValue("vy")*e.getIt();
df = Math.sqrt(dx*dx+dy*dy);
dx *= dr/df;
dy *= dr/df;
double nx = m.getdoubleValue("x") + dx;
double ny = m.getdoubleValue("y") +dy;
     nCoord[0] = nx;
     nCoord[1] = ny;
return nCoord;
    }
public double [] Freeze(Move e,Message m, Cell c){
double [] nCoord = new double[4];
nCoord[0] = m.getdoubleValue("x");
nCoord[1] = m.getdoubleValue("y");
nCoord[2] = 0;
nCoord[3] = 0;
return nCoord;
}
}
```

Code 13 Método `move` de la clase `runOut`.

```
...
public double [] move(Move e,Message m, Cell c){
  if(Glider.getTime() < 20)
    return e.move(m,c);
  else
    return NearestDoor.move(e,m,c);
}
...
```

Figura 4.2: Cuatro vistas de instantes de tiempo diferentes de la animación. El tiempo avanza de izquierda a derecha y de arriba a abajo.

Estas son algunas de las dinámicas posibles que pueden desarrollarse dentro de un espacio. El meta-modelo gSpaces ha sido creado para que los modelistas que lo utilicen puedan ajustarlo tanto como deseen a cada uno de los sistemas que esté modelando. Los agentes móviles de la librería poseen varias cualidades que permiten ajustar su comportamiento. Como ya hemos visto cada uno de los individuos posee una velocidad particular que puede ser cambiada, además cada agente móvil tiene un campo visibilidad puede utilizarse para modelar la dificultad para desalojar que puede tener un persona que no conoce el espacio donde se encuentra o también puede utilizarse en situaciones en las que, oscuridad o el humo disminuyan la visibilidad.

También hay información almacenada por las celdas como la cantidad de agentes móviles que la ocupan, la última dirección que fue seguida y la dirección preferencial que permiten modelar comportamiento en función de las acciones ejecutadas por los agentes móviles que se encuentran en el entorno inmediato. La descripción completa de los atributos de los agentes móviles y de las celdas se encuentran en la documentación de gSpaces.

trash in, trash out

Capítulo 5

Modelando con gSpaces usando AutoCAD

Existen varias formas de crear un `Space` en gSpaces. En esta sección haremos referencia a la forma que utiliza el traductor `dxf2g` el cual genera un esqueleto del modelo a partir de un archivo *Drawing Interchange Format* (DXF) de AutoCad®. La finalidad de esta sección es que el modelista conozca y comprenda el resultado generado por el compilador.

En este capítulo se explica como elaborar modelos con gSpaces. Comenzaremos con explicar como realizar y utilizar archivos .dxf para elaborar modelos con nuestra librería. Posteriormente se explican el conjunto de ejemplos que acompañan nuestra librería, que tienen diferentes configuraciones espaciales y diferentes reglas de movimiento. En este capítulo se describe todo el proceso de elaboración de un modelo, su simulación y finalmente se explican los resultados que arroja el simulador.

5.1. Usando AutoCAD como herramienta

Actualmente se utiliza herramientas de software para dibujar los diseños arquitectónicos y urbanos. Uno de los software difundidos en la comunidad de diseñadores es AutoCAD de Autodesk [10]. Dicho software arroja archivos en diferentes formatos como el *Data Exchange File* o dxf [17], que ha sido utilizado para crear nexos con otras herramientas. Puede almacenar información como objetos 3d, dimensiones, curvas, texto y otros en un formato sencillo. Para facilitar la creación de modelos con nuestra librería por personas relacionadas con el diseño se escogió este formato para realizar un traductor de modelos gSpaces. La librería gSpaces posee un traductor capaz de interpretar archivos en formato dxf y escribir modelos propios de nuestra librería.

Para que el traductor de la librería gSpaces pueda interpretar un conjunto de espacios dibujados en AutoCAD como un modelo gSpaces, el archivo que será guardado como `.dxf`, deberá poseer dos *layers* particulares que serán leídos por el traductor de gSpaces. En dichos *layers* serán dibujados los espacios y sus límites con ciertas características que permitirán al traductor crear el modelo. Las cualidades particulares que debe poseer el archivo son las siguientes:

- Poseer una capa o *layer* llamada gSpaces donde serán dibujados los límites no permeables, un punto interno y el nombre de cada uno de los espacios pertenecientes al modelo con un color diferente para cada espacio.
- Poseer una capa llamada `gDoors` donde se dibujarán los límites permeables pertenecientes a cada uno de los espacios con el color que corresponde al espacio al que pertenecen.
- Ser guardado como AutoCAD R14/LT98/LT97 DXF (*.dxf) o como AutoCAD 2000 / LT2000 DXF (*.dxf).

5.2. Dibujar un gSpace con AutoCAD.

Un espacio llevado a un modelo gSpaces como se explicaba en los capítulos 2 y 3 posee tres elementos fundamentales que son los límites no permeables, como por ejemplo las paredes, que en el meta-modelo se denominan `Walls`; los límites permeables, como las puertas, que en el meta-modelo se denominan `Doors` y el área contenida. Cada uno de estos elementos tiene una manera particular de ser dibujado en AutoCAD para que el traductor de la librería pueda convertirlos al modelo gSpaces.

5.2.1. El área contenida

El área contenida por el espacio debe ser señalada por medio de un punto dibujado con el comando *point* en la capa `gSpaces`. El punto debe estar ubicado dentro de los límites que envuelven el espacio y además su color tiene que ser el corresponde al espacio, como se muestra en la figura 5.1. debe estar completamente rodeada por límites permeables o no permeables.

Cada espacio poseerá un color particular, y todos los elementos que le pertenezcan serán dibujados en ese color, por lo tanto, los límites permeables, los no permeables, el nombre y el punto interno del un espacio deben dibujarse con un mismo color propio de ese espacio. Es muy importante que el área que posee el espacio esté totalmente contenida dentro de las líneas que lo definen, `Walls` y `Doors` y que sólo queden sin envolver los espacios a través de los cuales las personas no pueden circular.

También se pueden asignar los nombres para cada uno de los espacios modelados creando una línea de texto simple dentro de la capa, *layer* gSpaces con el nombre de cada uno de los

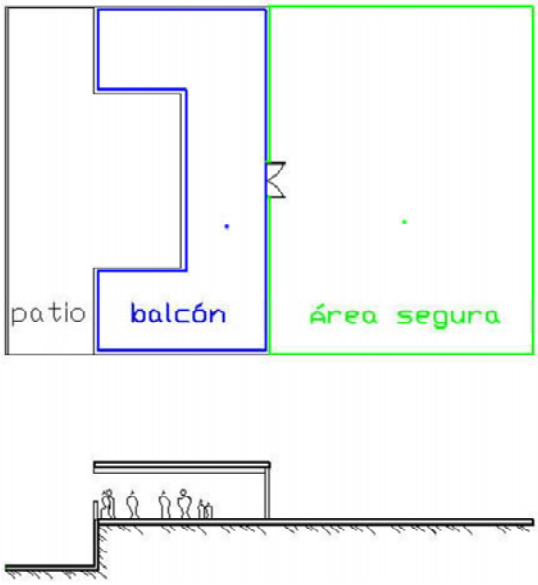

Figura 5.1: El modelo que se presenta en esta figura consta de tres espacios con diferentes alturas o niveles. En planta puede observarse como fueron creadas las diferencias entre los espacios como una de las posibles maneras de modelar. El espacio llamado **patio** no está comunicado por límites permeables con el espacio **balcón** porque para las personas no sería posible salvar la diferencia de nivel entre ambos espacios de manera segura. **balcón** está comunicado por medio de un límite permeable con **Área segura** que en este caso recibe ese nombre por ser el lugar que fue considerado más seguro dentro del modelo. Note que cada uno de los espacios está dibujado con un color particular y poseen un punto del mismo color para definir el área que está contenida dentro de sus límites. **patio** no posee límites de ningún tipo porque no contendrá personas dentro de su área y los límites de **balcón** que colindan con él son no permeables. El límite permeable que comunica **balcón** y **Área segura** está dibujado solo para **balcón**, en azul porque se trata de una `Door` unidireccional.

espacios en el color correspondiente. En caso de que no exista la línea de texto la librería les asignará nombres por defecto.

5.2.2. Los límites no permeables o `Walls`

Son dibujadas dentro de la capa o *layer* gSpaces con el comando `line` y deben encerrar, junto con los límites permeables, toda el área que estará contenida dentro del espacio. Las líneas deben ser dibujadas por todas las caras de las paredes que están en contacto con el área contenida como se muestra en la figura 5.2.

Note que las paredes que están en el perímetro del área contenida solamente se representan con una línea, mientras que las paredes internas están representadas por una línea por cada cara que está en contacto con en área.

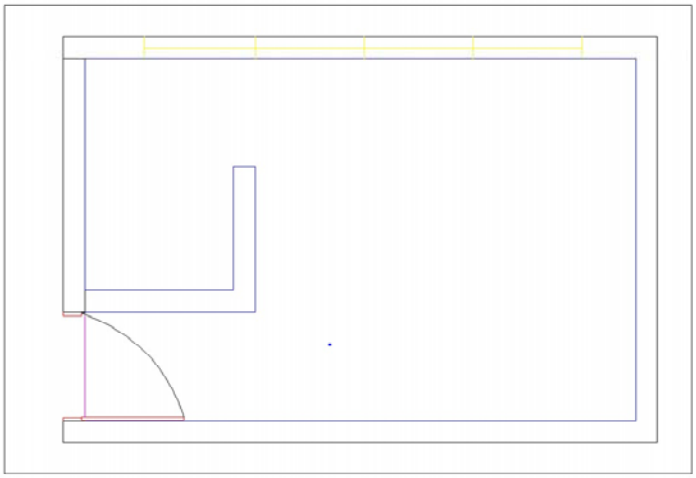

Figura 5.2: En la figura podemos observar un espacio sencillo con una puerta. Note que en azul están dibujadas las líneas que definen las caras de las paredes externas e internas que colindan con el área contenida. Para gSpaces las paredes externas son dibujadas con una línea con la forma de la caras internas de las paredes a diferencia de las internas que son dibujadas con líneas por todas sus caras. Dichas líneas definen los límites no permeables para el espacio dentro de la `capa` o `layer` `gSpaces` junto al punto que define el área contenida. La línea ubicada en la puerta que posee el mismo color correspondiente al espacio estará ubicada en la capa `gDoors` y define el límite permeable para el espacio.

5.2.3. Los límites permeables o `Doors`

Los límites permeables deben ser dibujados en la capa `gDoors` utilizando el el comando `line` y en el mismo color del espacio al que pertenecen (ver figuras 5.5 y 5.3). Un límite permeable pertenece al espacio del que los agentes móviles salen a través de ese límite y si mediante él se comunican dos espacios se tienen que dibujar dos líneas, una por cada uno de los espacios y cada una de las líneas con el color del espacio correspondiente siempre dentro del layer `gDoors`. Esta manera de crearlos brinda la posibilidad de tener estadísticas durante la simulación en cada sentidos del cruce del límite.

Las dos líneas que definen un límite bidireccional deben estar dibujadas exactamente en el mismo lugar, definidas por los mismos puntos y de colores diferentes, una por espacio. En caso de que las puertas no sean de doble sentido se debe dibujar una única línea del color correspondiente al espacio al que pertenece la puerta, es decir, el espacio desde el cual será utilizada como salida y el traductor la detectará como una puerta de tipo `'U'`. La manera como el traductor detecta y escribe el código de cada uno de los tipos de puertas fue explicada en el capítulo 3, en la sección 3.4. En el caso de las puertas que no comunican espacios y los individuos que las atraviesan salen del sistema, también deben ser dibujadas con una sola línea del color del espacio al que pertenecen.

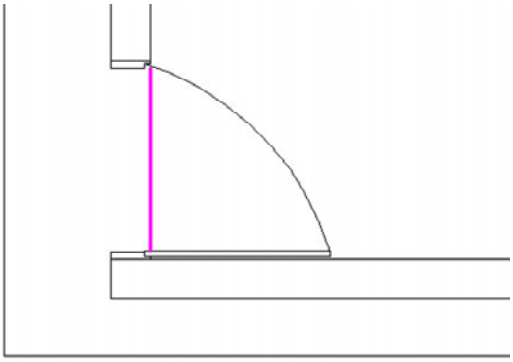

Figura 5.3: Forma como debe ser dibujada la puerta unidireccional con respecto al espacio en general.

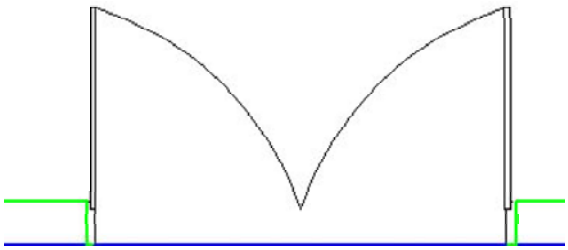

Figura 5.4: Detalle de las dos líneas que definen una puerta bidireccional.

5.3. Pasos a seguir para crear un modelo gSpaces

Utilizando AutoCAD como herramienta, los pasos para crear un modelo gSpaces, explicados de manera resumida, son los siguientes:

1. Conceptualizar los espacios en función del tipo de movimiento de los agentes móviles y de sus perspectiva (véase sección 2.2).

2. Crear un archivo de AutoCAD que posea dos capas, `gSpaces` y `gDoors`. Si se posee un archivo .dwg con el conjunto de espacios a modelar ya creado no es necesario crear uno nuevo, simplemente se deberán crear en este los layers `gSpaces` y `gDoors`.

3. Dibujar líneas envolventes utilizando el comando `line` con un color diferente para cada uno de los espacios dentro de la capa `gSpaces` si representan límites no permeables y en la capa `gDoors` si son permeables. Se debe recordar que las paredes internas del espacio les serán dibujadas todas sus caras. Si el archivo se ha creado especialmente para el modelo de gSpaces es conveniente dibujar los espacios utilizando las unidades de AutoCAD como equivalentes al sistema de medida que utiliza normalmente haciendo una relación uno a uno es decir, una unidad de AutoCAD equivaldrá a un

metro o pie, de esta manera el modelo puede quedar creado en escala uno (1) a uno (1) sin tener que hacer cambios en la resolución (véase sección 5.4).

4. Colocar en la capa `gSpaces`, dentro de cada una de las envolventes de los espacios, un punto en el color correspondiente, que indica el área contenida.

5. Asignar el nombre a cada uno de los espacios, por medio de una línea de texto sencilla del color correspondiente al espacio, dibujada dentro de la capa gSpaces.

Es muy importante que las capas `gSpaces`y `gDoors` estén construidas de manera correcta, sin errores en los nombres y que los elementos que se encuentran dentro de ellas estén correctamente dibujados, ya que de lo contrario, el traductor creará un modelo producto de lo que el modelista dibujó pero que no se corresponde con lo que desea modelar.

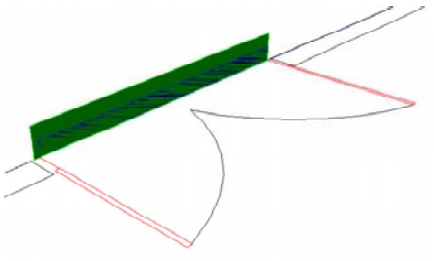

Figura 5.5: Detalle de las dos líneas que componen una puerta que comunica dos espacios. Note que existen dos líneas, una azul y otra verde a las cuales se les ha dado altura para facilitar su visualización.

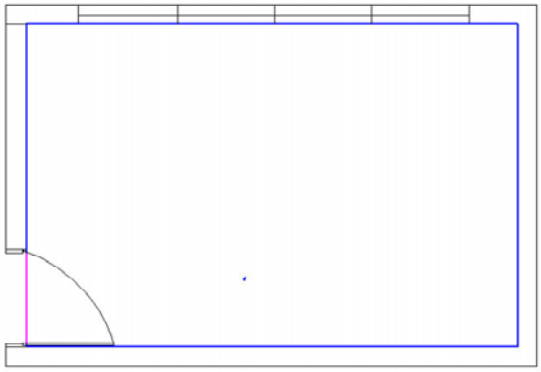

Figura 5.6: Ejemplo de un espacio y el punto que indica el área que contiene.

5.4. Resolución

AutoCAD posee unas unidades universales y cada usuario selecciona su equivalencia con respecto a la escala real. La librería gSpaces creará el modelo por omisión en estas

mismas unidades y cada celda tendrá como tamaño una de las unidades de AutoCAD; si, para quien dibuja las unidades de AutoCAD equivalen a un metro las celdas de los espacios del modelo tendrán como lado un metro. Esta equivalencia se debe tener muy clara porque si se desea que las celdas posean menor o mayor tamaño la resolución o lado de las celdas, deberá ser cambiado en los parámetros del espacio antes de realizar la simulación. En caso de que las unidades de AutoCAD en el dibujo equivalgan en el modelo a una medida diferente de uno, el modelista deberá adaptar la resolución del modelo al tamaño de las celdas que necesita, porque la librería por omisión colocará el mismo tamaño para las celdas. El tamaño de las celdas deberá ser el adecuado para los intereses del modelista con respecto al modelo. Por ejemplo, si el área comprendida dentro de una celda es la justa para que sólo pueda estar ocupada por un individuo será necesario considerar en la regla de movimiento el hecho de que dos individuos no pueden ocupar el mismo espacio y modelar el movimiento en función de esta determinante siempre y cuando el modelo lo requiera. Por el contrario, si el área es mayor y brinda la posibilidad de que dos o más individuos puedan ocupar una celda puede no ser necesario considerarlo. El tamaño de las celdas posee una estrecha relación con la velocidad que les será asignada a los individuos y la distancia que podrán recorrer en un instante de tiempo. Al discretizar un sistema que es continuo en espacio y tiempo se corre el riesgo de que el modelo pierda realismo de allí la importante relación entre el tiempo y la velocidad de los individuos, si la velocidad de un individuo le permite atravesar más de una celda en una actualización del modelo la influencia que la celda debía de tener sobre el no se cumple y el modelo pierde realismo. Para conservar la correspondencia del modelo con la realidad es importante determinar cual es la velocidad máxima de los individuos. La longitud de las celdas debe depender de la cantidad de pasos que, de acuerdo con la realidad, debe dar un determinado individuo para cambiar de celda y el tiempo en el que el sistema puede cambiar. Dicha cantidad de pasos dependerá de la velocidad máxima que posee cada individuo y las veces que es necesario que sea influenciado por las celdas. Por ejemplo si la velocidad máxima de los individuos es de 1.5 metros por segundo, las celdas poseen un tamaño de 3 metros cuadrados, y el modelo se actualiza cada segundo, el individuo será influenciado al menos dos veces por el entorno de la celda, en 30 segundos pueden ocurrir muchos eventos dentro del sistema que no llegarían a influenciarlo. El modelista debe conocer el modelo para decidir la frecuencia con la que debe aplicar la regla de movimiento, en función de los cambios del entorno para mantener la continuidad del modelo. Para esto se debe seleccionar un paso de simulación pequeño, por ejemplo si se utiliza 0.2seg el individuo, en cada actualización del modelo, podrá avanzar a lo sumo 30 centímetros lo que quiere decir que será influenciando por su entorno al menos 10 veces.

5.5. Ejecución del traductor

Una vez que ha sido construido el archivo .dxf se debe ejecutar el traductor para construir el modelo en código gSpaces. El traductor es el archivo que incluimos con nuestra librería llamado `dxf2g` que debe ser colocado en uno de los directorios de las variable de

ambiente `PATH`. La manera de ejecutar el traductor es muy sencilla, se debe ejecutar la instrucción,

`dxf2g` *archivo*`.dxf`

desde el directorio donde se encuentra el `dxf` a través de una consola de comandos. El comando `dxf2g` invoca al traductor mientras que *archivo* es el nombre del archivo con extensión `dxf` que contiene la descripción del espacio a modelar. El traductor generará un archivo `.java` con el nombre *archivo*`.java` dentro del mismo directorio donde se encuentra el archivo de origen. Dicho archivo contiene el modelo de simulación correspondiente al sistema dibujado en el archivo `.dxf`.

5.6. Modelos ejemplo de gSpaces

A continuación explicamos un conjunto de ejemplos con la finalidad de que el lector se familiarice con el funcionamiento de nuestra librería y conozca las capacidades de la misma. Presentamos algunas de las posibilidades que brinda gSpaces con respecto a la disposición del espacio, las reglas de movimiento y finalmente con respecto a cambios de la configuración de los espacios durante la simulación, es decir, en tiempo de ejecución.

5.6.1. Diferentes configuraciones espaciales

La librería posee flexibilidad en cuanto a la forma de los espacios que pertenecen a los modelos brindando la posibilidad de crear espacios cuyos límites posean formas variadas e irregulares que, además, pueden ser creadas en un archivo .dxf y traducidas por `dxf2g`. El ejemplo que presentamos en la figura 5.7 consiste en un espacio sencillo con una geometría regular y una salida. La actualización de las posiciones de los agentes móviles se realiza por medio de la regla de movimiento por omisión que posee nuestra librería. Dicha regla asigna una nueva posición de manera aleatoria a los agentes móviles con un paso constante.

Como puede observarse en el código 14 que se cita a continuación, el modelo posee los nodos `Display display, Node exit, MobileAg mobileAg y Space Room` y sus principales instrucciones para: construir el espacio `addWall, addDoor, build`, para llenar el espacio `mobileAg.addSpace`, para dibujar en la animación los espacios pertenecientes al modelo y sus dinámicas `display.addSpace`, iniciar la animación `display.setup` y los correspondientes al las variables manejadas por el simulador GALATEA como `Glider.setTitle, Glider.setTsim, GRnd.inisem, Glider.trace, Glider.stat, Glider.act y Glider .process`.

En el código 14 la puerta que es agregada al espacio es de `tipo U`, unidireccional, debido a que solo posee una dirección hacia la que pueden dirigirse los agentes móviles.

También pueden incluirse unos espacios dentro de otros, espacios compuestos, como es el caso que se observa en la figura 5.8 donde existe un espacio central que posee una puerta por donde los agentes móviles acceden a un segundo espacio que finalmente les permite acceder a la salida general del modelo.

Code 14 Código que simula el movimiento de agentes móviles de un espacio con una sola puerta.

```
/* @(#)Modelo.java 4 18/05/05 Kay Tucci
 * Copyright (c) Galatea Group. Universidad de Los Andes.
 */
package demos.desalojos;
import galatea.gspaces.*;
import galatea.glider.*;
/** @author Kay Tucci
 *
 * Codigo que simula el movimiento de agentes moviles de un espacios con una
 * sola puerta. Los agentes se mueven aleatoriamente dentro del espacio y si
 * alguno de ellos alcanza la puerta entonces abandona el espacio.
 */
public class OneRoomModel{
  /**OneRoomModel  -
   * Clase OneRoomModel constructor. The model is in this class
   */
  public OneRoomModel(){}
  /**Display node -
   * Display the spaces on screen. Second argument sets the display refreshment
   * @see galatea.gspaces.Display
   */
  public static Display display = new Display("Display",0.2);
  /**Node of general exit for the model
   * @see galatea.glider.Node
   */
  public static Node exit = new Node("Exit",'E');
  /**Node that creates the messages*/
  public static MobileAg mobileAg = new MobileAg("MobileAg",1.5,10);
  /**Room -
   * @see galatea.gspaces.Space
   */
  public static Space Room = new Space("Room", 10, 1.0, 1.0, 1.0, 1);
  public static void main(String args[]){
  /**Setting the Room*/
    Room.addWall(1.0, 0.0, 5.0, 0.0);
    Room.addWall(5.0, 0.0, 5.0, 4.0);
    Room.addWall(5.0, 4.0, 0.0, 4.0);
    Room.addWall(0.0, 4.0, 0.0, 0.0);
    Room.addDoor(0.0, 0.0, 1.0, 0.0, exit, 'U', 1, 0.0, 1.0);
    Room.build();
    mobileAg.addSpace(Room);           // Adding a mobil agent to Room
    display.addSpace(Room);            // Setting display spaces
    display.setup();
    Glider.setTitle("OneRoomModel");   // Setting the model title
    Glider.setTsim(10000);             // Setting simulation time
    GRnd.inisem();                     // Setting the random number generator
    Glider.trace("OneRoomModel.trc");  // Trace file name
    Glider.stat("OneRoomModel.sta");   // Statistics file name
    Glider.act(mobileAg,0);            // First model activations
    Glider.act(Room.getMove(),0);
    Glider.act(display,0);
    Glider.process();                  // Model processing
  }
}
```

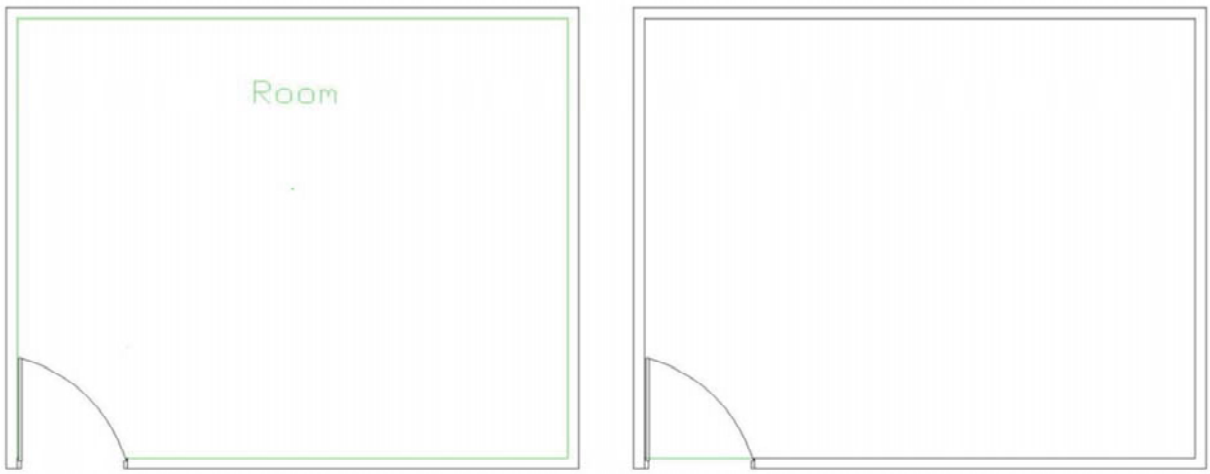

Figura 5.7: Modelo sencillo con un espacio y una puerta.

En el código 16 puede observarse que `Outside` posee como límites externos cuatro puertas unidireccionales que dirigen los mensajes hacia la salida general del modelo por medio de la instrucción que es arrojada por el traductor de gSpaces que se cita en el código 15.

Code 15 Puerta unidireccional que es agregada al espacio `Outside` en el modelo `RoomOut` codigo 16. Véase 3.4

```
Outside.addDoor(0.0, 0.0,  /*Xo,Yo*/
                    0.0, 8.0,   /*Xf,Yf*/
                    exit,   /*destiny node*/
                    'U',    /*type (U,B)*/
                    1,  /* ### capacity*/
                    0.0,    /* ### delay*/
                    1); /* ### selection coeficient*/
```

gSpaces también permite desarrollar modelos con espacios de geometrías complejas como el que se observa en la figura 5.9 cuyo código se encuentra junto al conjunto de demos que acompañan nuestra librería.

En el dibujo de la izquierda, correspondiente a la figura 5.9, note que los espacios que posean formas curvas o complejas, pueden ser dibujados por medio del comando `line` creando una forma semejante a las curvas por medio de varias líneas. Por otra parte, para crear paredes internas se deben dibujar todas las caras de las mismas como se había explicado anteriormente. Note que el espacio posee una escalera en forma de caracol que lo comunicará con un nivel superior que no está siendo modelado. La manera correcta de crear estos detalles del espacio es crear los límites de acuerdo a las áreas a través de las cuales pueden desplazarse los individuos. En este caso particular la escalera fue modelada

Code 16 Código que simula el movimiento de agentes móviles de un espacio con una sola puerta que comunica a los agentes a un espacio externo conformado por cuatro grandes Doors o límites permeables que, finalmente llevarán a los agentes a la salida general del modelo.

```
/* Mon Sep 05 13:19:27 GMT-04:00 2005
 * This code was generated automaticaly
 * from RoomOut.dxf AutoCad Dxf file by
 * Dxf2g Galatea tool
 - Copyright (c) Galatea Group. Universidad de Los Andes.
 */
package demos.desalojos;
import galatea.gspaces.*;
import galatea.glider.*;
/** @author Klaudia Laffaille
  * @version 0.1
 * Codigo que simula el movimiento de agentes moviles en tres
 * espacios que se encuentran comunicados por dos puertas.
 * Los agentes se mueven aleatoriamente dentro de los espacios
 *  y si alguno de ellos alcanza una de las puertas abandona el
 * espacio en el que se encuentra.
 */
public class RoomOut{
 /**Display node -
   *
   * Display the spaces on screen. Second argument sets the display refreshment
   * @see galatea.gspaces.Display */
  public static Display display = new Display("Display",1.0);
 /**Node of general exit for the model
   * @see galatea.glider.Node
   */
  public static Node exit = new Node("Exit",'E');
 /**Node that creates the messages*/
  public static MobileAg mobileAg  = new MobileAg("MobileAg",1.5,10);
 /**OutSide  -
  * @see gSpaces.Space
  */
  public static Space Outside = new Space("Outside",0,1.0,7.0,1.0,1.0);
 /**Room -
  * @see gSpaces.Space
  */
  public static Space Room = new Space("Room",0,3.0,5.0,1.0,1.0);
 /**RoomOut  -
   * Clase Model constructor. The model is in this class
   */
    public RoomOut(){
    }
```

Code 17 Continuación del código 16.

```
 public static void main(String args[]){
  /**Walls and Doors of space Outside*/
   Outside.addWall(1.9, 6.1,1.9, 1.9);
   Outside.addWall(8.1, 6.1,1.9, 6.1);
   Outside.addWall(8.1, 1.9,8.1, 6.1);
   Outside.addWall(1.9, 1.9,8.1, 1.9);
   Outside.addDoor(0.0, 0.0,0.0, 8.0,exit,'U',1,0.0,1);
   Outside.addDoor(10.0, 0.0,0.0, 0.0,exit,'U',1,0.0,1);
   Outside.addDoor(10.0, 8.0,10.0, 0.0, exit,'U',1,0.0,1);
   Outside.addDoor(0.0, 8.0,10.0, 8.0,exit,'U',1,0.0,1);
   Outside.build();
/**Walls and Doors of space Room*/
   Room.addWall(2.0, 6.0,2.0, 2.0);
   Room.addWall(8.0, 2.0,3.0, 2.0);
   Room.addWall(8.0, 6.0,8.0, 2.0);
   Room.addWall(2.0, 6.0,8.0, 6.0);
   Room.addDoor(3.0, 2.0,2.0, 2.0,exit,'U',1,0.0,1);
   Room.build();
  /**Spaces to be filled*/
   mobileAg.addSpace(Outside);
   mobileAg.addSpace(Room);
  /**Spaces to be displayed*/
   display.addSpace(Outside);
   display.addSpace(Room);
   display.setup();
   Glider.setTitle("RoomOut"); // Setting the model title
   Glider.setTsim(100); // Setting simulation time
   GRnd.inisem(); // Setting the random number generator
   Glider.trace("RoomOut.trc"); // Trace file name
   Glider.stat("RoomOut.sta"); // Statistics file name
   Glider.act(mobileAg,0);  // First model activations
   Glider.act(Outside.getMove(),0);
   Glider.act(Room.getMove(),0);
   Glider.act(display,0);
   Glider.process(); // Model processing
 }
}
```

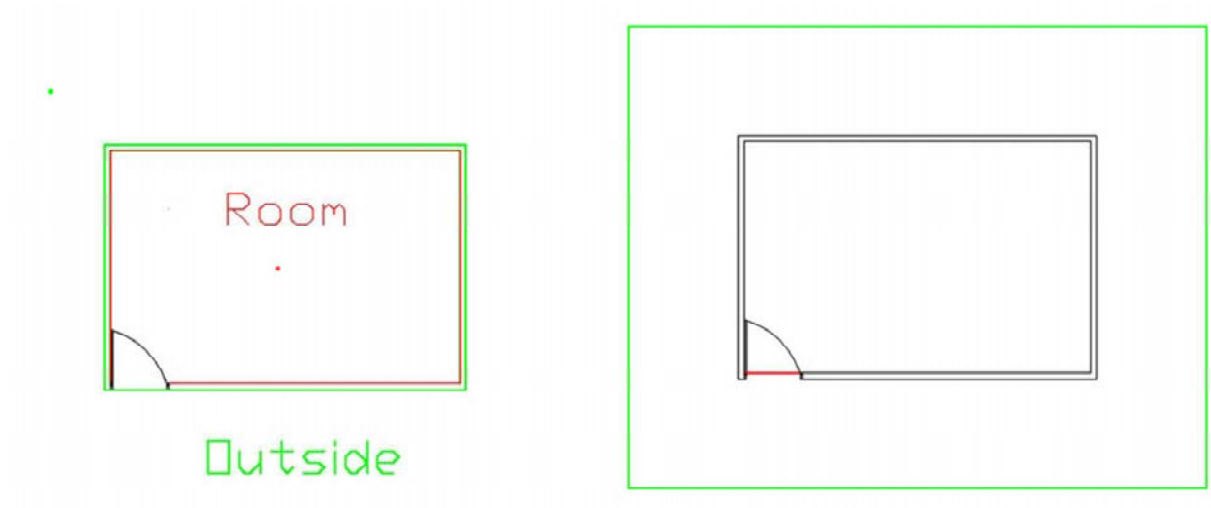

Figura 5.8: Espacios compuestos por otros espacios. El dibujo de la izquierda muestra los objetos contenidos en la capa `gSpaces` y el de la derecha los contenidos en la capa `gDoors`. El modelo presentando en la figura 5.8 está compuesto por dos espacios, `Out` contiene al espacio `Room` y define los límites permeables que llevarán a los agentes móviles hacia la salida general del modelo. `Outside` define el espacio externo en el que el comportamiento de los agentes es relevante para el modelo de acuerdo con las decisiones del modelista, una vez que los agentes móviles interceptan los límites permeables que pertenecen a `Out` salen del modelo. Note que el espacio llamado `Room` está comunicado con el espacio `Outside` por medio de una puerta unidireccional que se dibuja como una línea del color del espacio al que pertenece. Véase el código 16 correspondiente a la figura.

por medio de un límite permeable en el área donde se puede acceder a ella como se observa en el dibujo de la derecha de la figura 5.9 y junto a este límite fueron creados límites no permeables (izquierda en la figura) que definen el área en que, debido a la altura de la escalera, los agentes móviles no pueden transitar. Como el nivel hacia el que acceden los agentes no es pertinente para este modelo los agentes son dirigidos por la escalera hacia la salida general del modelo pero también pueden ser dirigidos a otro espacio si el modelista lo considera necesario. Véase el código 16 generado por el traductor para simular el comportamiento de los agentes móviles en el espacio correspondiente a la figura 5.9.

También pueden simularse modelos de mayor complejidad que incluyen puertas bidireccionales y espacios que se comunican en ambos sentidos a través de estas como el que se muestra en la figura 5.10.

La figura 5.10 muestra tres espacios que se encuentran comunicados por tres puertas. `LeftRoom` se comunica con `RightRoom` a través de la `Door` unidireccional que se encuentra en la pared central en la parte superior y `RightRoom` se comunica con `LeftRoom` a través de la puerta bidireccional que se encuentra en la parte inferior de la pared. `LeftRoom` se comunica también con `OutSide` a través de una puerta bidireccional ubicada en la esquina superior izquierda como se observa en la figura. Véase el código 20 que es el correspondiente

Code 18 Código que simula el movimiento de agentes móviles en un espacio de geometría compleja con una escalera de caracol y una puerta.

```
/* Mon Sep 05 12:54:53 GMT-04:00 2005
 * This code was generated automaticaly
 * from GeometricComplexModel.dxf AutoCad Dxf file by
 * Copyright (c) Galatea Group. Universidad de Los Andes.
 * Dxf2g Galatea tool
 */
package demos.desalojos;
import galatea.gspaces.*;
import galatea.glider.*;
/**@author Klaudia Laffaille
 * @version 0.1
 * Codigo que simula el movimiento de agentes moviles en un espacio de geometra compleja
 * con una escalera de caracol.  Los agentes se mueven aleatoriamente dentro del  espacio
 *  y si alguno de ellos alcanza una de las puertas o la escalera abandona el espacio.
 */
public class GeometricComplexModel{
 /**Display node -
   * Display the spaces on screen. Second argument sets the display refreshment
   * @see galatea.gspaces.Display
   */
  public static Display display = new Display("Display",1.0);
 /**Node of general exit for the model*/
  public static Node exit = new Node("Exit",'E');
 /**Node that creates the messages*/
  public static MobileAg mobileAg  = new MobileAg("MobileAg",1.5,10);
 /**ComplexSpace -
  * @see gSpaces.Space
  */
  public static Space ComplexSpace = new Space("ComplexSpace",0,10.04,
  4.015,1.0,1.0);
  /**GeometricComplexModel  -
   * Clase Model constructor. The model is in this class
   */
    public GeometricComplexModel(){
    }
```

Code 19 Continuación del código 18

```
    public static void main(String args[]){
   /**Walls and Doors of space ComplexSpace*/
    ComplexSpace.addWall(4.15, 6.2,4.15, 6.3);
    ComplexSpace.addWall(10.03, 4.01,10.11, 4.43);
    ComplexSpace.addWall(9.90, 3.53,10.04, 4.015);
    ComplexSpace.addWall(9.72, 3.07,9.90, 3.53);
  //Se continuan agregando paredes a ComplexSpace
    ComplexSpace.addWall(6.12, 8.76,7.62, 8.76);
    ComplexSpace.addDoor(6.31, 4.54,5.41, 4.49,exit, 'U',1,0.0,1);
    ComplexSpace.addDoor(6.11, 8.76,4.11, 8.76,exit,'U',1,0.0,1);
    ComplexSpace.build();
   /**Spaces to be filled*/
    mobileAg.addSpace(ComplexSpace);
   /**Spaces to be displayed*/
    display.addSpace(ComplexSpace);
    display.setup();
   Glider.setTitle("GeometricComplexModel"); // Setting the model title
   Glider.setTsim(100); // Setting simulation time
   GRnd.inisem(); // Setting the random number generator
   Glider.trace("GeometricComplexModel.trc"); // Trace file name
   Glider.stat("GeometricComplexModel.sta"); // Statistics file name
   Glider.act(mobileAg,0);
   Glider.act(ComplexSpace.getMove(),0);
   Glider.act(display,0);
   Glider.process(); // Model processing
  }
}
```

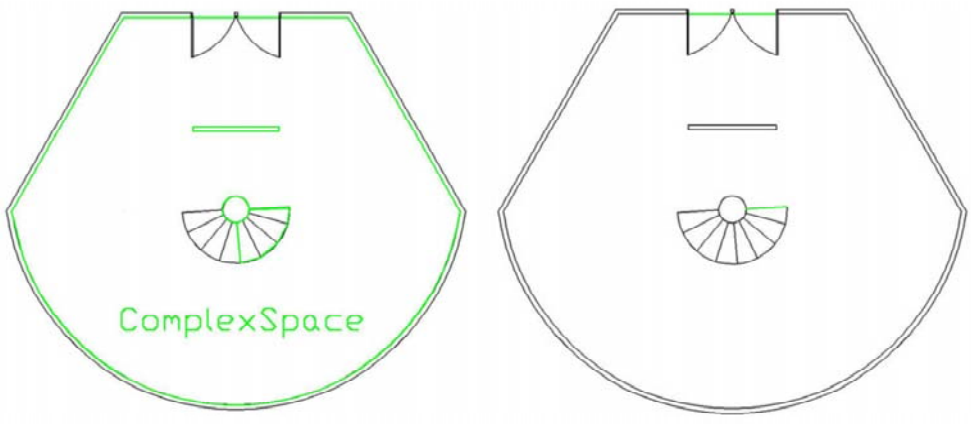

Figura 5.9: Modelo con un espacio de geometría compleja. El dibujo de la izquierda muestra los objetos contenidos en la capa `gSpaces` y el de la derecha los contenidos en la capa `gDoors`. Note que los objetos del dibujo de la derecha son los límites permeables, los puntos que definen el área contenida y los nombres de los espacios en los colores correspondientes. En la figura de la derecha pueden observarse los límites permeables correspondientes al espacio.

a la figura 5.10.

Los códigos correspondientes a los modelos presentados en las figuras anteriores se encuentran en el conjunto de demos que acompañan nuestra librería.

Code 20 Código que simula el movimiento de agentes móviles en espacios compuestos por otros espacios.

```
/*Tue Sep 06 11:13:56 GMT-04:00 2005
 * This code was generated automaticaly
 * from Model.dxf AutoCad Dxf file by
 * Dxf2g Galatea tool
 * Copyright (c) Galatea Group. Universidad de Los Andes.
 */
package demos.desalojos;
import galatea.gspaces.*;
import galatea.glider.*;
/* * @author Klaudia Laffaille
 * @version 0.1
 * Codigo que simula el movimiento de agentes moviles en tres espacios
 * que se encuentran comunicados por tres puertas. Los agentes se mueven
 * aleatoriamente dentro de los espacios y si alguno de ellos alcanza una
 * de las puertas abandona el espacio en el que se encuentra.
 */
public class Model{
  /**Display node -
   * Display the spaces on screen. Second argument sets the display refreshment
   * @see galatea.gspaces.Display
   */
  public static Display display = new Display("Display",1.0);
 /**Node of general exit for the model
   * @see galatea.glider.Node
   */
  public static Node exit = new Node("Exit",'E');
 /**Node that creates the messages*/
  public static MobileAg mobileAg  = new MobileAg("MobileAg",1.5,10);
 /**LeftRoom -
  * @see gSpaces.Space
  */
  public static Space LeftRoom = new Space("LeftRoom",0,4.0,4.0,1.0,1.0);
 /**RightRoom -
  * @see gSpaces.Space
  */
  public static Space RightRoom = new Space("RightRoom",0,10.0,6.0,1.0,1.0);
 /**OutSide -
  * @see gSpaces.Space
  */
  public static Space OutSide = new Space("OutSide",0,0.0,0.0,1.0,1.0);
 /**Model  -
   * Clase Model constructor. The model is in this class
   */
    public Model(){
    }
    public static void main(String args[]){
  /**Walls and Doors of space LeftRoom*/
    LeftRoom.addWall(8.0, 5.0,8.0, 10.0);
    LeftRoom.addWall(8.0, 2.0,8.0, 3.0);
    LeftRoom.addWall(8.0, 2.0,4.000000000000001, 2.0);
    LeftRoom.addWall(2.000000000000001, 10.0,8.0, 10.0);
    LeftRoom.addWall(2.000000000000001, 2.0,2.000000000000001, 10.0);
    LeftRoom.addDoor(8.0, 5.0,8.0, 3.0,RightRoom,'B',1,0.0,1);
    LeftRoom.addDoor(2.0, 2.0,4.0, 2.0,OutSide,'B',1,0.0,1);
    LeftRoom.build();
```

Code 21 Continuación del código 20

```
 /**Walls and Doors of space RightRoom*/
    RightRoom.addWall(8.0, 3.0,8.0, 2.0);
    RightRoom.addWall(8.0, 7.0,8.0, 5.0);
    RightRoom.addWall(8.0, 10.0,8.0, 9.0);
    RightRoom.addWall(14.0, 10.0,8.0, 10.0);
    RightRoom.addWall(14.0, 2.0,14.0, 10.0);
    RightRoom.addWall(8.0, 2.0,14.0, 2.0);
    RightRoom.addDoor(8.0, 9.0,8.0, 7.0,exit,'U',1,0.0,1);
    RightRoom.build();
 /**Walls and Doors of space OutSide*/
    OutSide.addDoor(16.0, -3.6000001401139195,0.0, 0.0,exit,'U',1,0.0,1);
    OutSide.addDoor(16.0, 12.0,16.0, -3.6000001401139195, exit,'U',1,0.0,1);
    OutSide.addDoor(0.0, 12.0,16.0, 12.0,exit,'U',1,0.0,1);
    OutSide.addDoor(0.0, 0.0,0.0, 12.0,exit,'U',1,0.0,1);
    OutSide.build();
   /**Spaces to be filled*/
   mobileAg.addSpace(LeftRoom);          // Adding a mobil agent to LeftRoom
   mobileAg.addSpace(RightRoom);        // Adding a mobil agent to RightRoom
   mobileAg.addSpace(OutSide); // Adding a mobil agent to OutSide
   /**Spaces to be displayed*/
    display.addSpace(LeftRoom);
    display.addSpace(RightRoom);
    display.addSpace(OutSide);
    display.setup();
    Glider.setTitle("Model"); // Setting the model title
    Glider.setTsim(100); // Setting simulation time
    GRnd.inisem(); // Setting the random number generator
    Glider.trace("Model.trc");  // Trace file name
    Glider.stat("Model.sta");  // Statistics file name
    Glider.act(mobileAg,0);  // First model activations
    Glider.act(LeftRoom.getMove(),0);
    Glider.act(RightRoom.getMove(),0);
Glider.act(OutSide.getMove(),0);
    Glider.act(display,0);
    Glider.process(); // Model processing
  }
}
```

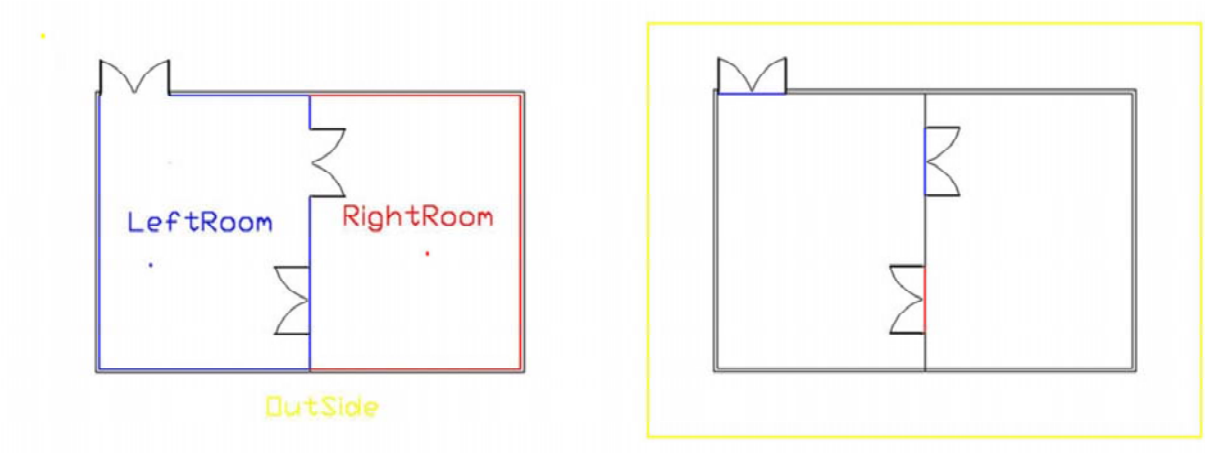

Figura 5.10: Espacios compuestos por otros espacios. El dibujo de la izquierda muestra los objetos contenidos en la capa `gSpaces` y el de la derecha los contenidos en la capa `gDoors`. Note en el dibujo de la derecha que las `Doors` bidireccionales son dibujadas con dos líneas ubicadas exactamente entre los mismos puntos y de colores diferentes, los correspondientes a los espacios que comunica la `Door`.

Referencias

[1] Biham O., Middleton A.A. y Levine D., *Self-organization and dynamical transition in traffic-flow models.*Phys. Rev. A, **46** (1992)

[2] d'Inverno M., Luck M. (2001) *Understanding Agent Systems.* 2nd rev. and extended ed., 2004, XVIII, 240 p. Hardcover.

[3] Domingo C.(1995). *Proyecto GLIDER.*, e-122-92. informe 1992- 1995. Reporte, CDCHT, Universidad de Los Andes. Mérida. Venezuela.

[4] Guevara T. (1.996). *La responsabilidad Multidisciplinaria en el Diseño Urbano y de Edificaciones en Zonas de Actividad Sísmica.*, Curso de Actualización. Fundación Pablo Miliani, Centro de Ingenieros del Estado Trujillo. Valera Edo. Trujillo, Venezuela.

[5] Nagel K. y Herrmann H.J., *Deterministic models for traffic jams.* Physica A, **199** (1993)

[6] Norberg-Schulz C. *Existencia, Espacio y Arquitectura.* Editorial Blume, 1975 Edición original: EXISTENCE, SPACE AND ARCHITECTURE, Ed. Studio Vista, Londres.

[7] Uzcátegui M.*Diseño de la plataforma de simulación de GALATEA.*, Tesis de Maestría, Maestría en Computación, Facultad de Ingeniería, Universidad de Los Andes. Mérida. Venezuela.

[8] Wolf D.E., Schreckenberg M., Bachem A., editores *Traffic and Granular Flow.* World Scientific. (1996)

REFERENCIAS WEB

[9] Ansgar, K., Andreas, S. (2002). *Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics.*, Institut fur Theoretische Physik, Universitat zu Koln, D-50923 Koln, Germany. arXiv.org:cond-mat/0203461

[10] Autodesk. *Estados Unidos*
http://usa.autodesk.com/adsk/servlet/home?siteID=123112&id=129446

[11] ARQHYS.com *Definiciones de Espacio.*
http://www.arqhys.com/definiciones-espacio-arquitectura.html

[12] ARIES, (2003) *ARIES*
http://archives.aries-ing.com/industrial/acustica_vibra/sim_sismic.htm

[13] Cells Alive *Chemotaxis*,
http://www.cellsalive.com/chemotx.htm.

[14] Consejo de Preservación y Desarrollo de la Universidad Central de Venezuela. *Universidad Central de Venezuela, Caracas Venzuela.*
http://146.155.99.60/arquitectura/patrimonio/html/psem_001_a.html

[15] Domingo, C., Colaboradores y Asesores. 2003/Versión en inglés: 2001. *Proyecto CDCHT I-524-95-02-AA Universidad de Los Andes. Mérida, Venezuela.*
http://afrodita.faces.ula.ve/Glider/manual4/index.html

[16] Escobar, R., y de la Rosa, A. (2003). *Architectural Design for the Survival Optimization of Panicking Fleeing Victims.*,Chalmers University of Technology, Complex Adaptive Systems Masters Program 412 96 Gothenburg, Sweden
http://www.dd.chalmers.se/ armando/panic/report.htm

[17] Internet FAQ Archives, Online Education. *DXF - Autodesk Drawing eXchange Format*
http://www.faqs.org/faqs/graphics/fileformats-faq/part3/section-45.html

[18] Dávila, J. (2000) *Netriders (agentes inteligentes).*Centro de Simulación y Modelos (CESIMO), Facultad de Ingeniería, Universidad de los Andes. Mérida Venezuela.
http://webdelprofesor.ula.ve/ingenieria/jacinto jacinto@ula.ve

[19] Dávila, J. *Inteligencia Artificial, capítulo 2.*,Centro de Simulación y Modelos (CESIMO) Universidad de los Andes. Mérida Venezuela.
http://webdelprofesor.ula.ve/ingenieria/jacinto/ia/capítulo02.pdf jacinto@ula.ve.

[20] Figueroa, J. (2003) *Urbanismo Preventivo enfocado para Santiago Chile.*,
http://urbanismo.8m.com/lecturas.htm

[21] Gutierres, B. (2001) *El Espacio Arquitectónico.* TODOARQUITECTURA.COM.
http://www.todoarquitectura.com/

[22] Helbing, Farkas y Vicseck, (2000). *Simulating dinamical features of escape panic.*, Colegium Budapest, Hungary, Institute of Economics and Traffic, Dresden University of technology Dresden Germany. arXiv.org/0009448

[23] Hoeger, B., (2002). *Teoría de la Simulación.* Guías de clase del curso de postgrado de Simulación y Modelado de Sistemas. Centro de Simulación y Modelos (CESIMO). Universidad de los Andes, Facultad de Ingeniería. Mérida Venezuela. hhoheger@ula.ve de la guía

[24] Laffaille, K. (2002). *Lineamientos Espaciales Urbanos para Disminuir la Vulnerabilidad Sísmica en Áreas Destinadas para Nuevos Desarrollos.* Tesis de Pre-grado. Universidad de los Andes, Facultad de arquitectura y Arte. Mérida Venezuela.

[25] Laffaille, K. (2004). *Incendio en las torres más altas de Caracas.*, http://www.todoarquitectura.com/v2/noticias/one_news.asp?IDNews=2116

[26] Laffaille, K. (2005). *Despues de las lluvias en Venezuela...* Noticia publicada para Todoarquitectura.com.
http://www.todoarquitectura.com/v2/noticias/one_news.asp?IDNews=2355

[27] Portal Parlamentario del Perú y el Mundo. *Leyes de las Indias, Libro Cuarto* Congreso de la República del Perú. Palacio Legislativo - Plaza Bolívar. Av. Abancay. Perú.
http://www.congreso.gob.pe/ntley/libroIndia.asp?wLibro=Cuarto

[28] Aguilar, T. Maria, V. (2002). *Como aparecieron las Ciudades del Nuevo Mundo.* Editado por: Asociación Cultural Odiseo, C/ Obsidiana, 6 10 6B2 29006 Málaga (España), 2001-2002.
http://usuarios.lycos.es/odiseomalaga/mo_04.htm

[29] Safina, M., (2003). *Vulnerabilidad sísmica de edificaciones esenciales. Análisis de su contribución al riesgo sísmico.* capítulo 13 Departamento de Ingeniería del Terreno, Cartografía y Geofísica. UCP. Salvador.
http://www.tdx.cesca.es/TDX-0225103-164824/

[30] SIGA *Amenazas* Secretariado de Manejo del Medio Ambiente para América Latína y el Caribe. Montevideo, Uruguay.
http://www.ems-sema.org/siga/siga/html/conceptos/a.html

[31] Tapia, F., Aranguren, R., de la Herrán, M. *Gaia en redcientifica.com*, http://www.redcientifica.com/gaia/ac/auto_c.htm

[32] Uzcátegui, M., Dávila, J., Tucci, k. (2004). *Simulación Multi-Agente con GALATEA.*, Centro de Simulación y Modelos (CESIMO). Facultad de Ingeniería, Universidad de los Andes. Mérida Venezuela.
http://cesimo.ing.ula.ve/INVESTIGACION/PROYECTOS/GALATEA/GALWEB/organizamarcos.htm

Apéndice A

Modelo de un sector de la ciudad universitaria de la UCV

A continuación describimos el modelo llamado `demoUCV` que acompaña a nuestra librería. La herramienta fue aplicada en uno de los accesos de mayor tránsito de la Ciudad Universitaria de Caracas (puerta 3), en dos edificaciones que están ubicadas junto a dicho acceso. Como lo muestra la figura A.1, las edificaciones se encuentran a la derecha del

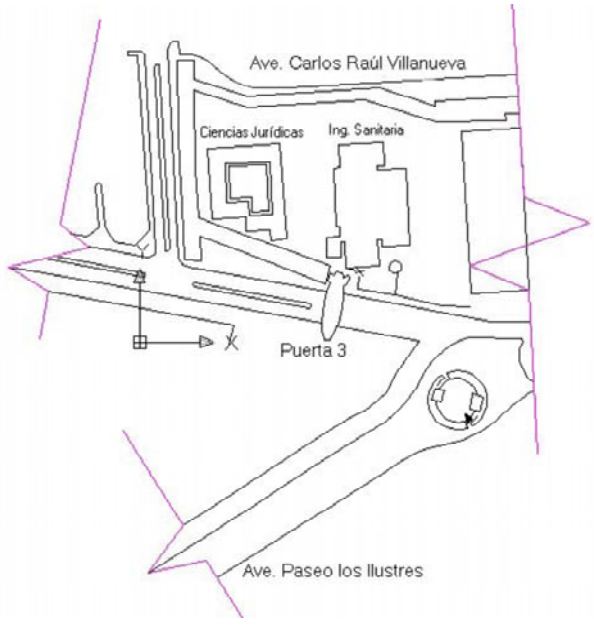

Figura A.1: Vista aérea de la puerta 3 de la Ciudad Universitaria de Caracas.

acceso vehicular del *Paseo los Ilustres*, y corresponden al edificio de Ingeniería Sanitaria, al que llamaremos `Edificio1`; y al Postgrado de Ciencias Jurídicas, identificado en el modelo como `Edificio2`.

Con la finalidad de mostrar la versatilidad del modelado con gSpace, cada una de las edificaciones es modelada de manera particular. Para el `Edificio1` no se desea estudiar la dinámica interna generada por los agentes móviles al desplazarse, por lo que el retraso que puedan sufrir los agentes dentro de la edificación es modelado como mediante una disminución en la velocidad promedio de los agentes. En la segunda edificación si es estudiada la dinámica interna de los agentes al desalojarla. Esta edificación posee un patio central que es modelado como un espacio a parte hacia el que se dirigen todos los individuos que se encuentran a su alrededor. Ambas edificaciones desalojan a un área verde despejada que puede que considerarse como el *área segura* dentro del modelo. Teniendo en cuenta lo anterior, para el modelo se definieron 4 espacios como se muestra en la figura A.2, donde

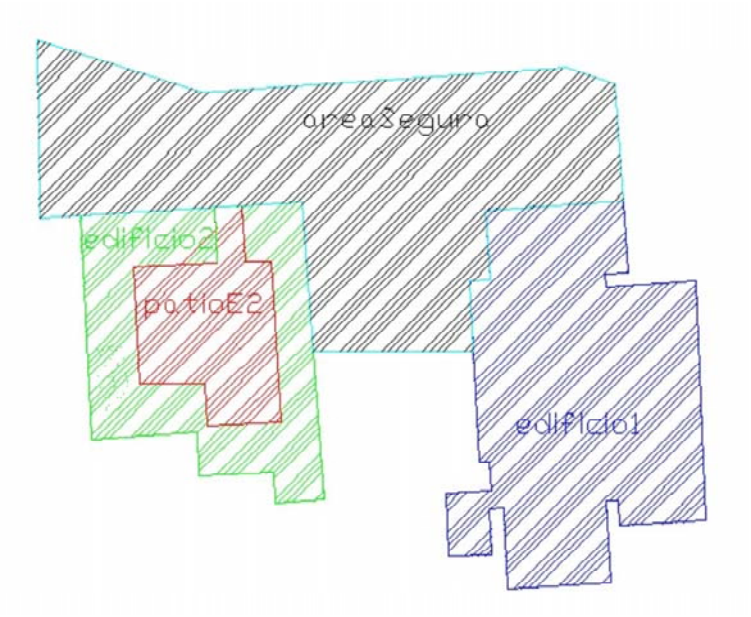

Figura A.2: Definición de los 4 espacios del modelo de la puerta 3 de la Ciudad Universitaria de Caracas.

al `Edificio1` le corresponde un único espacio, la edificación del Postgrado de Ciencias Jurídicas lo conforman dos espacio: `PasilloE2` y `PatioE2`; y por último, tenemos el espacio correspondiente al *área segura* al que denominaremos dentro del modelo `AreaSegura`. El código 22-25 describe el modelo correspondiente a este pequeño sector de la Ciudad Universitaria.

Code 22 Código del modelo para un sector de la Ciudad Universitaria de Caracas.

```
/*
 * Sat Apr 30 22:15:49 GMT-04:00 2005
 *
 * This code was generated automatically from UCVs.dxf
 * AutoCad Dxf file by Galatea Dxf2g tool
 */
package demos.desalojos;
import galatea.gspaces.*;
import galatea.glider.*;
/**
 *
 * Modelo de desalojo de los edificios adyacentes a la puerta 3 de la
 * Ciudad Universitaria de Caracas
 *
 * @author AUTOR
 * @version VERSION
 */
public class UCVs{
 /**Display node -
  *
  * Display the spaces on screen.
  * @see gSpaces.Display
  */
  public static Display display = new Display("Display",1.0);
 /**Node of general exit for the model*/
  public static Node exit = new Node("Exit",'E');
 /**Node that creates the messages*/
  public static MobileAg creator = new MobileAg("Creator",1.5,20);
//  Declaration of nodes related with Edificio1
 /**Edificio1 -
  * @see gSpaces.Space
  */
  public static Space Edificio1 = new Space("Edificio1",  /*Space Name*/
                10,                   /* ### Initial number of people*/
                67.03188514595644,  /*internal point x coordinate*/
                34.81073097823902,  /*internal point y coordinate*/
                1.0,                  /* ### space resolution*/
                1.0,                  /* ### time step*/
                "Uc.RuleUCV");
//  Declaration of nodes related with PatioE2
 /**PatioE2 -
  * @see gSpaces.Space
  */
  public static Space PatioE2 = new Space("PatioE2",  /*Space Name*/
                0,                    /* ### Initial number of people*/
                23.8828414576398,   /*internal point x coordinate*/
                35.1071730329171,   /*internal point y coordinate*/
                1.0,                  /* ### space resolution*/
                1.0,                  /* ### time step*/
                "Uc.RuleUCV");
//  Declaration of nodes related with PasilloE2
 /**PasilloE2 -
  * @see gSpaces.Space
  */
  public static Space PasilloE2 = new Space("PasilloE2",  /*Space Name*/
                0,                    /* ### Initial number of people*/
                25.0,                 /*internal point x coordinate*/
                45.0,                 /*internal point y coordinate*/
                1.0,                  /* ### space resolution*/
                1.0);                 /* ### time step*/
```

Code 23 Continuación del modelo del código 22.

```
//  Declaration of nodes related with AreaSegura
 /**AreaSegura -
  * @see gSpaces.Space
  */
  public static Space AreaSegura = new Space("AreaSegura",  /*Space Name*/
                0,                  /* ### Initial number of people*/
                45.13603595550005,  /*internal point x coordinate*/
                52.72825958794999,  /*internal point y coordinate*/
                1.0,                /* ### space resolution*/
                1.0);               /* ### time step*/
  public UCVs(){}
  public static void main(String args[]){
   /**Walls of space Edificio1*/
    Edificio1.addWall(85.61121551991965, 50.85728338340536,     /*Xo,Yo*/
                      76.62325402283693, 50.55094209987864);    /*Xf,Yf*/
    Edificio1.addWall(55.76442349568496, 49.83999992254202,     /*Xo,Yo*/
                      55.12212373546134, 0.7388738677201446)    /*Xf,Yf*/
    Edificio1.addWall(87.5718546298695, 2.273572525025818,      /*Xo,Yo*/
                      55.12212373546134, 0.7388738677201446);   /*Xf,Yf*/
    Edificio1.addWall(85.61121551991965, 50.85728338340536,     /*Xo,Yo*/
                      87.5718546298695, 2.273572525025818);     /*Xf,Yf*/
   /**Doors of space Edificio1*/
    Edificio1.addDoor(76.62325286865234, 50.550941467285156,    /*Xo,Yo*/
                      55.76442337036133, 49.84000015258789,     /*Xf,Yf*/
                      AreaSegura,       /*destiny node*/
                      'U',              /*type (U,B)*/
                      1,                /* ### capacity*/
                      0.0,              /* ### delay*/
                      1);               /* ### selection coeficient*/
   /**Grid construction of Edificio1*/
    Edificio1.build();
   /**Walls of space PatioE2*/
    PatioE2.addWall(30.66932451880791, 42.92699761157637,       /*Xo,Yo*/
                    27.30400200952789, 42.82057199921498);      /*Xf,Yf*/
    PatioE2.addWall(31.83247083520198, 21.8421197781088,        /*Xo,Yo*/
                    30.66932451880791, 42.92699761157637);      /*Xf,Yf*/
    PatioE2.addWall(22.2596810181785, 21.53767785966358,        /*Xo,Yo*/
                    31.83247083520198, 21.8421197781088);       /*Xf,Yf*/
    PatioE2.addWall(21.94172516994109, 27.30527311656977,       /*Xo,Yo*/
                    22.2596810181785, 21.53767785966358);       /*Xf,Yf*/
    PatioE2.addWall(13.21098309898212, 26.8771979637179,        /*Xo,Yo*/
                    21.94172516994109, 27.30527311656977);      /*Xf,Yf*/
    PatioE2.addWall(12.62119067595582, 42.35623992315001,       /*Xo,Yo*/
                    13.21098309898212, 26.8771979637179);       /*Xf,Yf*/
    PatioE2.addWall(23.56787196056563, 42.70241989723184,       /*Xo,Yo*/
                    12.62119067595582, 42.35623992315001);      /*Xf,Yf*/
    PatioE2.addWall(27.30400200952789, 42.82057199921498,       /*Xo,Yo*/
                    26.76888003476122, 50.14286042321185);      /*Xf,Yf*/
    PatioE2.addWall(23.29447896223917, 49.8775912410681,        /*Xo,Yo*/
                    23.56787196056563, 42.70241989723184);      /*Xf,Yf*/
   /**Doors of space PatioE2*/
    PatioE2.addDoor(27.30400276184082, 42.82057189941406,       /*Xo,Yo*/
                    23.56787109375, 42.70241928100586,          /*Xf,Yf*/
                    PasilloE2,      /*destiny node*/
                    'U',            /*type (U,B)*/
                    1,              /* ### capacity*/
                    0.0,            /* ### delay*/
                    1);             /* ### selection coeficient*/
   /**Grid construction of PatioE2*/
    PatioE2.build();
```

Code 24 Continuación del modelo del código 22.

```
/**Walls of space PasilloE2*/
 PasilloE2.addWall(26.76892916829148, 50.14286417453792,      /*Xo,Yo*/
                   27.30533654571237, 42.82061420284636);     /*Xf,Yf*/
 PasilloE2.addWall(23.29364472504638, 49.89948571106689,      /*Xo,Yo*/
                   26.76892916829148, 50.14286417453792);     /*Xf,Yf*/
 PasilloE2.addWall(23.56787196056563, 42.70241989723184,      /*Xo,Yo*/
                   23.29364472504638, 49.89948571106689);     /*Xf,Yf*/
/**Doors of space PasilloE2*/
 PasilloE2.addDoor(26.76892852783203, 50.14286422729492,      /*Xo,Yo*/
                   23.29281234741211, 49.8994255065918,       /*Xf,Yf*/
                   AreaSegura,        /*destiny node*/
                   'U',               /*type (U,B)*/
                   1,                 /* ### capacity*/
                   0.0,               /* ### delay*/
                   1);                /* ### selection coeficient*/
/**Grid construction of PasilloE2*/
 PasilloE2.build();
/**Walls of space AreaSegura*/
 AreaSegura.addWall(76.62325402283693, 50.55094209987864,     /*Xo,Yo*/
                    75.47249472977369, 66.34003651793086);    /*Xf,Yf*/
 AreaSegura.addWall(55.76442349568496, 49.83999992254202,     /*Xo,Yo*/
                    76.62325402283693, 50.55094209987864);    /*Xf,Yf*/
 AreaSegura.addWall(55.12212373546134, 31.73887386772014,     /*Xo,Yo*/
                    55.76442349568496, 49.83999992254202);    /*Xf,Yf*/
 AreaSegura.addWall(36.07500007649053, 31.28999992254192,     /*Xo,Yo*/
                    55.12212373546134, 31.73887386772014);    /*Xf,Yf*/
 AreaSegura.addWall(34.59000007649041, 50.73999992254186,     /*Xo,Yo*/
                    36.07500007649053, 31.28999992254192);    /*Xf,Yf*/
 AreaSegura.addWall(0.3200000764904338, 48.579999922542,      /*Xo,Yo*/
                    34.59000007649041, 50.73999992254186);    /*Xf,Yf*/
 AreaSegura.addWall(7.64902737E-8, 71.91999992254193,         /*Xo,Yo*/
                    0.3200000764904338, 48.579999922542);     /*Xf,Yf*/
 AreaSegura.addWall(21.61000007649084, 65.1999999225419,      /*Xo,Yo*/
                    7.64902737E-8, 71.91999992254193);        /*Xf,Yf*/
 AreaSegura.addWall(75.47249472977369, 66.34003651793086,     /*Xo,Yo*/
                    21.61000007649084, 65.1999999225419);     /*Xf,Yf*/
/**Doors of space AreaSegura*/
 AreaSegura.addDoor(21.610000610351562, 65.19999694824219,    /*Xo,Yo*/
                    75.47249603271484, 66.34003448486328,     /*Xf,Yf*/
                    exit,              /*destiny node*/
                    'U',               /*type (U,B)*/
                    1,                 /* ### capacity*/
                    0.0,               /* ### delay*/
                    1);                /* ### selection coeficient*/
 AreaSegura.addDoor(7.649027367051531E-8, 71.91999816894531,  /*Xo,Yo*/
                    21.610000610351562, 65.19999694824219,    /*Xf,Yf*/
                    exit,              /*destiny node*/
                    'U',               /*type (U,B)*/
                    1,                 /* ### capacity*/
                    0.0,               /* ### delay*/
                    1);                /* ### selection coeficient*/
/**Grid construction of AreaSegura*/
 AreaSegura.build();
```

Code 25 Continuación del modelo del código 22.

```
    /**Spaces to be filled*/
     MobileAg.addSpace(Edificio1,10,1);  /* Add 10 type 1 mobil Ags to Edificio1 */
     MobileAg.addSpace(AreaSegura,20,1); /* Add 20 type 1 mobil Ags to AreaSegura */
     MobileAg.addSpace(PatioE2,10,1);    /* Add 10 type 1 mobil Ags to PatioE2 */
     MobileAg.addSpace(PasilloE2,40,1);  /* Add 40 type 1 mobil Ags to PasilloE2 */
    /**Spaces to be displayed*/
     display.addSpace(Edificio1);
     display.addSpace(AreaSegura);
     display.addSpace(PatioE2);
     display.addSpace(PasilloE2);
    /**Animation setup*/
     display.setup();
    /**Model title*/
     Glider.setTitle("UCVs");
    /**Simulation time*/
     Glider.setTsim(100);                  /* ### 100 is the default value*/
    /**Random seed initialization*/
     GRnd.inisem();
    /**Set trace file name*/
     Glider.trace("UCVs.trc");
    /**Set statictic file name*/
     Glider.stat("UCVs.sta");
    /**Creator node activation*/
     Glider.act(creator,0);                /* ### 0 is the default value*/
    /**Edificio1 movement node activation*/
     Glider.act(Edificio1.getMove(),0);    /* ### 0 is the default value*/
    /**AreaSegura movement node activation*/
     Glider.act(AreaSegura.getMove(),0);   /* ### 0 is the default value*/
    /**PatioE2 movement node activation*/
     Glider.act(PatioE2.getMove(),0);      /* ### 0 is the default value*/
    /**PasilloE2 movement node activation*/
     Glider.act(PasilloE2.getMove(),0);    /* ### 0 is the default value*/
    /**Display node activation*/
     Glider.act(display,0);                /* ### 0 is the default value*/
    /**Network processing*/
     Glider.process();
  }
}
```