TECNOLÓGICO DE MONTERREY

Campus Ciudad de México

Escuela de Graduados en Ingeniería y Arquitectura

Maestría en Ciencias de la Computación

"Modelo de Simulador Distribuido de Línea de Ensamble para Uso Didáctico Accesible desde Dispositivos Móviles"

Autor:

Santiago García Alba Garciadiego

Supervisor de Tesis:

Dr. Víctor Manuel de la Cueva Hernández

Índice

Índice de Figuras

Índice de Tablas

Índice de Código

Capítulo 1

Introducción

En este capítulo se da una introducción al proyecto de esta tesis, al problema que se pretende resolver así como una justificación de la realización de dicho proyecto, y una descripción de los objetivos y alcances que se pretenden cumplir.

1.1. Planteamiento del Problema de Investigación

El que las universidades e institutos donde se imparte la carrera de Ingeniería Industrial y Sistemas, cuenten con celdas de manufactura de las cuales los alumnos puedan probar su funcionamiento, su capacidad para abastecer diferentes tipos de demanda con diferentes parámetros, y el impacto en su desempeño al establecer diferentes programas de producción, pueden ser de gran utilidad para que dichos alumnos refuercen sus conocimientos de temas como Pronósticos o Administración de Inventarios. Sin embargo, debido al elevado costo de adquisición de dichos equipos, generalmente el acceso que los alumnos tienen a estos es limitado en cuanto a tiempo (al tener que compartirlos con muchos alumnos). Además, en ocasiones las celdas de manufactura son difíciles de configurar o permiten realizar solamente un número limitado de modificaciones a su configuración. Por otro lado, resultaría muy complicado que los estudiantes pudieran probar diferentes distribuciones de demanda o de llegada de productos no terminados, más aún a largo plazo, lo cual es indispensable para observar el desempeño de la línea al tratar de satisfacer cualquier distribución de demanda no constante.

Debido a lo mencionado anteriormente, el complementar las celdas de manufactura para uso didáctico con un simulador, de manera que el alumno pueda experimentar más tiempo con el sistema al hacer uso de una celda de manufactura virtual, representa una posible solución al problema del acceso limitado que los alumnos suelen tener a dichas celdas de manufactura. Además, un simulador permitiría al alumno realizar pruebas sobre periodos largos de tiempo y

en un mayor número de escenarios (e.g. escenarios con diferentes demandas) y llevar un registro exacto del desempeño de la celda (posiblemente en una base de datos) a partir del cual se puedan obtener diversas gráficas e indicadores, que permitan al usuario realizar un mejor análisis de los tiempos de producción y los diferentes tamaños de inventario.

Adicionalmente, el simulador podría mejorar o complementar la experimentación que se haga de la celda de manufactura con diferentes aplicaciones y herramientas de software que, por ejemplo, permitan a los alumnos utilizar el simulador de forma distribuida, para realizar prácticas colaborativas, analizar cada módulo del sistema por separado, y observar el desarrollo de las simulaciones a diferentes escalas de tiempo, además de tener acceso a dicho simulador a través de internet, ya sea utilizando una computadora o un dispositivo móvil, de manera que el acceso y la experiencia de aprendizaje mejoren al brindar al usuario mayor flexibilidad en cuanto a dónde y cuándo utilizar el simulador. Además, dicha aplicación podría proporcionar herramientas como un generador de planes de producción, que determine planes de producción óptimos de acuerdo a la demanda, o algún asistente inteligente que analice el desempeño del alumno y le de retroalimentación para mejorarlo.

1.2. Justificación del Problema de Investigación

El proyecto presenta oportunidades tanto didácticas como económicas. En el ámbito didáctico, como se mencionó en la sección anterior, el simulador ayudaría a los alumnos a hacer un mejor uso a la celda de manufactura, obteniendo más práctica, y teniendo una mejor comprensión y una mayor capacidad para analizar la celda y proponer mejoras. Por otro lado, el uso del simulador en forma colaborativa y por módulo, permitiría la creación de nuevos tipos de prácticas colaborativas que brinden una mejor experiencia de aprendizaje. Adicionalmente, el alumno podría accesar al simulador fácilmente en forma remota.

Desde el punto de vista económico se evitaría incurrir en los costos por el uso de licencia que se tendrían que cubrir al usar algún paquete de simulación comercial. Además, es sabido que dichos paquetes tienden a disminuir la velocidad de ejecución de las simulaciones y a hacer un uso relativamente amplio de recursos de la computadora [L'Ecuyer, 2005], lo cual es especialmente indeseable si se va a utilizar el simulador en línea, si se quiere que este sea utilizado por medio de dispositivos móviles, los cuales tienen menores capacidades que las computadoras "convencionales", si se quiere que este ejecute simulaciones en tiempo real o si se van a procesar grandes cantidades de datos.

El acceso a la aplicación tiene gran importancia, el poder permitir a los alumnos acceder desde sitios remotos a través de laptops o PCs les permite tener una mayor flexibilidad en su estilo y horarios de trabajo, al tiempo que ejercitan habilidades relativas al uso de la tecnología y para la realización de actividades colaborativas. El poder utilizar la aplicación desde dispositivos móviles da al usuario una flexibilidad aun mayor al brindarle la opción de trabajar en movimiento y en un mayor número de ambientes posibles, como pueden ser viajes, medios de transporte o situaciones donde no se puede disponer de una computadora portátil (e.g. cuando estén en el sitio donde realicen el servicio social).

La educación móvil[1], aprovechando las crecientes capacidades de imagen y sonido de los celulares, representa una importante oportunidad de brindar al alumno la flexibilidad de acceder a contenidos didácticos en casi cualquier momento y casi cualquier lugar, lo que permite mayor personalización en la experiencia de aprendizaje, al tiempo que se fortalecen diversas habilidades profesionales, como el uso de tecnología de punta, la comunicación efectiva y el trabajo en equipo. En algunas instituciones educativas, como es el caso del Tecnológico de Monterrey, los alumnos tienen acceso a varios servicios desde dispositivos móviles (e.g. acceso a biblioteca),

[1] Sheng, H, Siau, K, Fui-Hoon Nah, F 2010, 'Understanding the values of mobile technology in education: a value-focused thinking approach', *ACM SIGMIS Database*, vol. 41, Issue 2, ACM New York, NY, USA p. 25, consultado en ACM Digital Library.

además de la posibilidad de interactuar entre sí y realizar consultas por medio de dichos dispositivos.

También es importante considerar el uso de aplicaciones multiusuario [Patterson, 1991], lo cual permite la realización de prácticas colaborativas que fortalezcan la experiencia de aprendizaje, al impulsar el intercambio de ideas y conocimientos entre los alumnos. Además, es necesario brindar la posibilidad de crear un simulador que pueda integrar otro tipo de servicios al usuario, como el uso de tutores inteligentes, aplicaciones que generen planes de producción óptimos, entre otros.

Por otro lado, las aplicaciones de simulación gráficas comerciales que pudieran ser disponibles para los alumnos carecen de acceso web, de capacidad de uso desde dispositivos móviles y de herramientas de programación prácticas para integrar otro tipo de funcionalidades al simulador como son el uso de herramientas adicionales de programación o la integración con aplicaciones que, por ejemplo, generen planes de producción óptimos en base a condiciones determinadas por el usuario.

Es importante mencionar también que los paquetes de simulación comerciales carecen de facilidades para representar aspectos complejos, en algunos casos relacionados con manufactura flexible, como son el uso de diferentes combinaciones de artículos en los lotes de salida, y la sincronización entre los diferentes elementos de las celdas de manufactura flexible, tanto para la implementación de paradigmas de producción, como son el de "*push*" o el de "*pull*", así como para el control de colisiones y la coordinación entre las diversas máquinas automáticas que componen las celdas de manufactura automatizadas.

1.3. Definición de Objetivos

Para brindar al usuario una experiencia similar a la que obtendría al usar la línea de ensamble, y para que los análisis que obtenga del simulador sean aplicables al sistema real, es necesario que dicho simulador represente con precisión el funcionamiento de la línea de producción real, de manera que pueda complementar el uso de ésta adecuadamente. De igual modo, es importante proporcionar al alumno herramientas que lo guíen en el uso de la aplicación, de modo que se aproveche el programa de *software* para ayudar al usuario a reforzar sus conocimientos relacionados al uso de éste, y que lo apoyen en el análisis del sistema, ya sea por medio de indicadores de desempeño o por medio de herramientas automáticas de optimización.

La accesibilidad de la aplicación es un punto importante a considerar. Es necesario brindar al usuario flexibilidad en cuanto a lugar y tiempo para el uso de la aplicación. Será importante además proporcionar la posibilidad de integrar el simulador en ambientes de *e-learning*[2], los cuales brindan un medio flexible de comunicación y apoyo, así como herramientas para realizar actividades colaborativas.

Para aprovechar las ventajas de la implementación de aplicaciones de uso didáctico para dispositivos móviles, mencionadas en la sección anterior, y para seguir las tendencias hacia el uso de dispositivos móviles en el ámbito académico, el permitir acceso al simulador desde este tipo de tecnología representa una característica importante para el simulador a desarrollar.

Existen así mismo tendencias hacia el aprendizaje colaborativo, buscando que los alumnos, al trabajar en grupos, intercambien información y puntos de vista, de manera que, al tener los integrantes las mismas metas en equipo, se fortalezca el aprendizaje individual de cada uno de

[2] "El e-learning, o aprendizaje electrónico puede ser visto como un enfoque de innovación para proporcionar ambientes de aprendizaje bien diseñados, interactivos, y facilitados para el uso de cualquiera, en cualquier lugar y en cualquier momento al utilizar los atributos y recursos de varias tecnologías junto con otras formas de material didáctico adecuadas para ambientes de aprendizaje abiertos, flexibles y distribuidos" [Kahn 2005, p.3].

ellos mediante la interacción con sus compañeros. Al mismo tiempo, con este tipo de aprendizaje se pretende que igualmente desarrollen habilidades de tipo social, como liderazgo, comunicación y solución de conflictos. En base a lo anterior, la posibilidad de generar prácticas de tipo colaborativo representa otra característica a agregar al simulador.

En base a lo expuesto los objetivos del presente trabajo quedan definidos de la siguiente manera:

- Desarrollar un modelo para la creación de simuladores de líneas de producción que representen con un amplio grado de precisión el sistema que van a modelar y que puedan ejecutar simulaciones en tiempo real.
- Desarrollar un modelo complementario para implementar el simulador de manera que permita un amplio y fácil acceso, pudiendo ser utilizado desde dispositivos móviles y pudiendo ser integrado en un ambiente de *e-learning*, además de permitir la creación de prácticas colaborativas.
- Utilizar el modelo propuesta para desarrollar el simulador de una línea de ensamble de uso didáctico.
- Implementar el modelo de manera que pueda ser accesible desde dispositivos móviles y que permita la creación de prácticas colaborativas (e.g. por medio de sesiones multiusuario).

1.4. Alcance de la Investigación

De acuerdo con los objetivos establecidos para el presente trabajo, al finalizar éste se contará con lo siguiente:

- Un modelo para el desarrollo de simuladores de líneas de producción que tengan un alto grado de precisión y que puedan ser ejecutados en tiempo real.

- Un modelo para implementar el simulador de líneas de ensamble de modo que sea accesible a través de Internet desde dispositivos móviles y permita la creación de prácticas colaborativas, por medio de sesiones multiusuario.

El simulador a desarrollar deberá ser capaz de tener un impacto en la capacidad de los alumnos para analizar la una celda de manufactura de uso didáctico y para generar propuestas para mejorar el desempeño de esta. A partir de esto, se espera que los alumnos cuenten con una herramienta adicional para adquirir, por medio de la práctica, conocimientos más sólidos en temas como pronósticos, Administración de Inventarios, Planeación de la Producción y, principalmente, el uso de simulación para la el diseño y la administración de sistemas.

El modelo desarrollado, por su parte, deberá ser sólido y fácil de usar para el desarrollo de simuladores de este tipo, además deberá estar acompañado de código que pueda ser fácilmente reutilizable para la generación de aplicaciones nuevas, que modelen sistemas similares.

El simulador creado sentaría las bases para que en un futuro sean agregadas una presentación gráfica más elaborada de este, una herramienta para exportar su configuración a la línea de ensamble, de manera que el simulador sirva adicionalmente como una interfaz amigable para el uso de la celda de manufactura y, probablemente, un generador automático de planes de producción y un asistente inteligente.

1.5. Organización del Documento

Capítulo 2: Antecedentes

En este capítulo se da una descripción del uso de simulación tanto en el campo de la Ingeniería Industrial como en el académico, así como una descripción del estado del arte en herramientas de

simulación usadas en la práctica y la enseñanza de la Ingeniería Industrial, y en el desarrollo de simuladores de ese tipo.

Adicionalmente se describe el estado del arte en otros temas adicionales relacionados con el desarrollo del proyecto, que son los simuladores distribuidos y las aplicaciones móviles.

Capítulo 3: Modelo para Simuladores Móviles

Se comienza con una descripción general de la estructura del simulador, explicando de forma breve los dos principales bloques en los que la aplicación se dividió para su desarrollo, que son la funcionalidad del simulador y el desarrollo del simulador como sistema distribuido.

Después se da una descripción del sistema que se pretende simular, que es la celda de manufactura para uso didáctico del laboratorio de Ingeniería Industrial del Tecnológico de Monterrey. Posteriormente se modela dicho sistema como un modelo de colas.

Se describen las características del simulador móvil distribuido y el uso que se le va a dar a este. Se da una descripción de los requerimientos que la aplicación deberá cubrir, así como la forma en que el usuario hará uso de la aplicación y el tipo de características especiales que ofrecerá la aplicación. También, se da una descripción general de cómo es que funcionará el simulador móvil distribuido.

Posteriormente se da una descripción detallada de los dos principales bloques que componen el simulador, incluyendo la estructura que se le dio a cada uno de estos y los elementos que los componen, así como una justificación de los patrones y arquitecturas seguidas, y las herramientas utilizadas.

Capítulo 4: Implementación

Aquí se describe a detalle la programación de la funcionalidad del simulador y las diferentes clases usadas para ello, así como la implementación en código del simulador como aplicación distribuida accesible desde dispositivos móviles. Se describe cómo se implementó el simulador para ser utilizado desde dispositivos BlackBerry, y cómo es que se levantó el servicio Web mediante el cual se integraron los diferentes componentes del simulador móvil distribuido.

Capítulo 5: Resultados

Se describen tanto las pruebas realizadas al simulador móvil distribuido como el desempeño de la aplicación en dichas pruebas. Igualmente se habla de las implicaciones que tuvo el modelo en el proceso de desarrollo de la aplicación.

Capítulo 6.2: Conclusiones y Trabajo a Futuro

Se mencionan observaciones y conclusiones que se obtuvieron en relación al modelo creado, señalando ventajas y desventajas encontradas en éste. Del mismo modo se dan observaciones acerca de del simulador implementado y el uso que se le podría dar.

Posteriormente se dan sugerencias de posibles trabajos a futuro que se podrían realizar en base a, o en relación a, este proyecto, que incluyen mejoras en alguno de sus aspectos, integración de herramientas adicionales y el desarrollo de nuevas aplicaciones que utilicen conceptos y herramientas similares a los usados en el presente proyecto.

1.6. Proceso de Investigación Seguido

El periodo de investigación constó de tres partes principales:

1) Investigación del estado del arte
2) Análisis de la línea de ensamble
3) Desarrollo de software.
4) Implementación de la aplicación

La **Investigación del Estado del Arte** consistió en la recopilación de información relevante que ayudara en la elección de los tipos de herramientas y técnicas a utilizar, así como a determinar las posibles características a agregar al simulador de manera que se cumpla con los objetivos del proyecto. Del mismo modo se buscó información acerca de proyectos similares que pudiera servir para delinear las características a agregar al simulador o posibles métodos de desarrollo a seguir.

El análisis de la línea de ensamble comprendió la modelación del sistema y la obtención de sus parámetros significativos. Al ser extenso y poco documentado el código de control de la celda de ensamble, se tuvo que realizar un análisis del sistema por medio de observación directa, de ese modo se determinó cuál es el comportamiento de la celda de ensamble y los tiempos implicados en cada una de las tareas que en ella se realizan.

Para analizar el funcionamiento y los tiempos del sistema se hizo funcionar la celda bajo los posibles escenarios que pudieran implicar un comportamiento distinto de la celda de ensamble, para de esa manera determinar cuáles eran las reglas bajo las cual operaba ésta. Para determinar los tiempos de la máquina se utilizó un cronómetro. Cada vez que se tenían determinadas las reglas bajo las cuales operaba cada módulo y los tiempos necesarios para modelar el proceso que éste llevaba a cabo, se realizaba un modelo de dicho módulo para después integrarlo al simulador.

En la parte de **desarrollo de software** primero se realizó la programación de la funcionalidad del simulador, el cual se creó de manera modular, implementando los diferentes componentes que modelan los diferentes módulos de la línea de ensamble uno por uno. Posteriormente se desarrolló la programación de los elementos del sistema que permitirían a los usuarios utilizar el simulador de manera remota y multiusuario desde dispositivos móviles.

La parte de **Implementación de la aplicación** consistió en la puesta en marcha de la aplicación en dispositivos móviles BlackBerry 8130, para esta parte del proyecto se utilizaron los dispositivos BlackBerry pertenecientes al laboratorio de redes del Tecnológico de Monterrey campus ciudad de México, así como el servidor del Departamento de Ingeniería y Arquitectura de la misma institución.

Capítulo 2:

Antecedentes

Este capítulo ofrece un panorama acerca del uso que se le da a los simuladores en la disciplina de Ingeniería Industrial, así como una descripción de las herramientas que actualmente existen para el desarrollo de dicho tipo de aplicaciones. Por otro lado, este capítulo también habla de los métodos y tecnologías para la implementación de simuladores distribuidos y para dispositivos móviles.

2.1. Uso de Simuladores como Herramientas Didácticas

Además de tener importancia por su papel en la práctica para el análisis de plantas de manufactura o cualquier clase de proceso de negocio, los ambientes de simulación son herramientas poderosas para apoyar el aprendizaje ya que animan al estudiante a la exploración, permitiéndole manipular parámetros del sistema simulado y visualizar los resultados de dichas acciones. Además permiten el entrenamiento de alumnos sin los pormenores que resultarían en el equipo al cometer errores propios de un principiante. En el mundo académico los simuladores son utilizados para complementar y mejorar la experiencia de aprendizaje obtenida en las clases y los laboratorios, resultando eficaces para involucrar al alumno en los temas tratados. [Rose, 2000]

De acuerdo con [Pang, 1999], la tecnología computacional provee un nuevo paradigma al apoyar al enfoque de aprendizaje por experiencia (*learning by doing*) de varias maneras:

- La simulación por computadora permite a los estudiantes la realización de tareas que en la vida real son muy costosas o muy peligrosas.
- Los simuladores pueden dar retroalimentación para asesorar al alumno. Las actividades de simulación por computadora guiadas pueden ser usadas como una alternativa para

motivar a los alumnos a que desarrollen habilidades para el descubrimiento por cuenta propia y de razonamiento.

La simulación brinda herramientas para ayudar al alumno a llevar a cabo una preparación estructurada de casos de estudio, lo cual resulta conveniente pues se ha encontrado que los alumnos tienden a relacionarse mejor con experiencias personales que con abstracciones.

2.2. Uso de Simulación para Sistemas de Manufactura

De acuerdo con [Schulze, 2000], los modelos de simulación para sistemas de manufactura pueden dividirse en tres categorías generales de acuerdo con su uso:

- Modelos de para diseño de sistemas.
- Modelos para administración de sistemas (estos son conocidos como modelos de simulación administrativa, MSM por sus siglas en inglés).
- Modelos para entrenamiento de personal.

La simulación es ampliamente usada en la operación cotidiana de plantas de producción para evaluar el desempeño y la capacidad de esos sistemas al existir cambios en diversos aspectos de producción, como pueden ser cambios en las condiciones o en los equipos de trabajo, sin embargo el mayor uso que recibe proviene de la administración, que utiliza dichos programas para detectar cuellos de botella, analizar los flujos de trabajo y evaluar decisiones operativas en relación con el desempeño del mismo sistema. Podría decirse que lo que la administración busca por medio de la simulación es, en cierto modo, "experiencia del futuro"[3].

[3] Schulze, T & Schumann, M 2000, 'Based Simulation Models as Management Tools for Assembly Lines', *Proceedings of the 32nd Winter Simulation Conference*, Winter Simulation Conference, p.1393, consultado en Abril de 2009 en ACM Digital Library.

La simulación de las líneas de producción es usada para resolver preguntas del tipo "*que pasa si...*", haciendo posible observar el comportamiento y los resultados al utilizar diferentes configuraciones de estas, u observar el impacto de determinadas condiciones de trabajo tanto controlables como no controlables. El poder simular las líneas de producción permite visualizar los posibles problemas que pudieran ser resultado de ajustes en cualquiera de los equipos que componen la línea de producción, por ejemplo, un par de brazos mecánicos podrían realizar sus labores en una línea de ensamble exitosamente durante años y sin embargo chocar al existir un ligero cambio en la trayectoria de uno de ellos para adaptarse a un nuevo diseño de producto. [VanDoren, 2003]

En general resulta muy difícil poder prever todos los posibles escenarios en los que una línea de producción puede encontrarse. La única manera en la que se pudieran observar todos los posibles fallos de manera eficaz es probando el funcionamiento de la línea de producción de manera experimental, sin embargo, esto trae consigo pérdidas por la alteración del funcionamiento de dicha línea de ensamble (al existir pérdida de tiempo de producción) y posibles pérdidas por daño de equipo, más los costos incurridos al realizar las modificaciones. Por esta razón, la simulación del sistema de control y de la línea de producción representa una buena opción para revisar el funcionamiento del sistema, dada la distribución del equipo en la línea antes de implementar cualquier cambio o diseño en el mundo real. [VanDoren, 2003]

Como ejemplo se pude considerar el caso de [Taj, 1998] en el cual se utilizó una herramienta de simulación llamada LABOR para probar una propuesta de rediseño de una planta de componentes de una compañía automotriz que consistía en cambiar el arreglo ya existente, en el cual las máquinas estaban ordenadas de manera departamental (agrupadas por tipo de operación) a uno de tipo celular, en el cual las máquinas necesarias para producir una pieza determinada estén cerca la una de la otra.

Con dicho simulador se analizaron diversos aspectos como la eficacia en la producción, los flujos de proceso, los tiempos para realizar los cambios de herramienta necesarios, los tiempos muertos de las máquinas, además de los tiempos utilizados para mantenimiento y reparaciones. Como resultado se pudo observar que, debido a que no todas las máquinas podían utilizarse para producir algunos tipos de pieza y que los tiempos para ajustar las máquinas para la producción de los diferentes tipos de pieza eran demasiado largos, era necesario realizar una inversión considerable para la mejora de dichas máquinas para realizar los cambios propuestos. Además pudieron observar otro tipo de detalles, como el hecho de que algunas de las máquinas eran muy poco confiables en su funcionamiento, inclusive se pudo observar también la existencia de máquinas demasiado grandes que provocaban que los trabajadores gastaran demasiado tiempo en recorrerlas de un extremo a otro. [Taj, 1998]

En [Fischer, 2005] se ha desarrollado un simulador multiusuario para permitir el análisis de sistemas de manufactura en grupos de trabajo. Dicho sistema permite a los usuarios interactuar conectados en línea en un ambiente virtual en 3D que representa el sistema a modelar, a través de este sistema controlan la simulación y observan los resultados de ésta. Para la implementación del simulador se utilizó un servidor que se encargaría exclusivamente de la ejecución de la simulación y un servidor dedicado a manejar la interacción entre los usuarios, así como tareas necesarias que no forman parte de la ejecución de la simulación, como la comunicación entre usuarios, planeación de movimiento, entre otras, dejando la tarea de la representación y la interfaz de usuario a la computadora del usuario, de este modo se le permite a cada usuario tener la libertad de observar el sistema de acuerdo a sus necesidades, independientemente de las de sus compañeros de trabajo.

2.3. Uso de Simuladores como Herramientas de Entrenamiento de Operadores

El entrenamiento de personal de una línea de producción es comúnmente realizado con equipo que busque asemejarse al equipo de control real de la planta de producción, de manera que mientras los operarios más experimentados trabajen en la línea de verdad, los nuevos puedan entrenarse con controladores y líneas virtuales, que de acuerdo con [VanDorean, 2003], si el

simulador es lo suficientemente realista, los trabajadores no notarán la diferencia y estarán listos para trabajar con el equipo real.

El uso de juegos de video es una alternativa para implantar un sistema de tipo "*learning by doing*", de acuerdo con [Van der Zee, 2005], la creación de un videojuego requiere de implementar el simulador usando lenguajes de simulación en lugar de herramientas de simulación gráficas basadas en librerías de bloques de construcción estándar dependientes de parámetros, debido a que estas últimas no brindan la flexibilidad requerida por los videojuegos para lidiar con tomadores de dediciones humanos.

[Van der Zee, 2005] considera aspectos adicionales a integrar en la creación de un ambiente de juego como son la retroalimentación y el control de las dediciones tomadas por el usuario, así como de los momentos en que estas se realizan y de las opciones o posibles acciones a realizar presentes en ese momento, además de otros aspectos como el manejo de excepciones por acciones inválidas, retroalimentación, análisis de desempeño del usuario y personalización del ambiente de juego. De esta manera con los videojuegos se puede lograr una mayor inmersión del usuario al tener éste que reaccionar ante diferentes eventos y escenarios generados en tiempo de ejecución.

2.4. Uso de Simuladores como Herramientas Didácticas en el Campo de la Ingeniería Industrial

En [Schultz, 2005] se propone un enfoque de enseñanza del uso de simulación basado en multimedia y casos, que pretende evitar un supuesto problema observado por dicho autor que radica en la falta de capacidad de los alumnos para proponer mejoras a modelos de sistemas existentes, a pesar de tener conocimientos sólidos tanto teóricos como prácticos acerca de la creación de modelos, esto probablemente por la falta de experiencia.

El enfoque mencionado anteriormente se compone de 3 módulos principales:

- **Páginas de hipertexto:** las cuales dan una introducción general al uso y navegación de la aplicación así como una introducción y una descripción del proceso modelado y los diversos equipos involucrados en él, adicionalmente se proporcionan diferentes soluciones con información relativa a los costos y su impacto en el funcionamiento del sistema.
- **Streaming Video:** un video que muestra el funcionamiento del proceso real.
- **Modelo de simulación:** Se proporciona un modelo con el fin de que este sea modificado y probado por los alumnos, quienes deben posteriormente hacer proposiciones de mejora y justificarlas.

2.5. Simuladores Gráficos para el Aprendizaje de Ingeniería Industrial

Hoy en día existe gran cantidad de software en el mercado que permite crear simuladores específicos fácil y rápidamente por medio de interfaces gráficas, como es el caso de Arena y Extend. A dichos ambientes de programación se les conoce con el nombre de "*point and click*"[4]. Algunas de estas herramientas de modelación, como es el caso de Extend [Krahl, 2003] permiten importar y exportar datos desde y hacia otros programas, de hecho, Extend, aparte de proporcionar dichas facilidades, es de "código abierto", lo cual permite pensar tanto en la posibilidad de modificar su código como en la de importar y exportar datos entre Extend y aplicaciones de simulación especialmente creadas para el proyecto. Las herramientas *pick and click* ofrecen, además, la seguridad de hacer uso del conocimiento de expertos en análisis y simulación de procesos involucrados en el desarrollo de la herramienta en cuestión[5].

En el caso de Extend, se puede utilizar su herramienta de modelación gráfica para algunas partes de la simulación y sólo modificar el código de los elementos que sea necesario, esto podría permitir concentrar un mayor esfuerzo del proyecto en otras posibles áreas de desarrollo, como, por ejemplo, el desarrollo de interfaces de usuario realistas que se asemejen a las interfaces del

[4] Schulze, T & Schumann, M. (2000), 'Language Based Simulation Models as Management Tools for Assembly Lines', *Proceedings of the 32nd Winter Simulation Conference*, Winter Simulation Conference, p.1393, consultado en Abril de 2009 en ACM Digital Library.

[5] <www.arenasimulation.com>

sistema a modelar, o la implementación de herramientas de apoyo, como pudieran ser asistentes inteligentes, que guíen al usuario en el uso de la simulación.

2.5.1. Principales Ejemplos de herramientas de Simulación Gráficas comerciales

A continuación se hace una breve descripción de los principales ambientes de tipo *pick and click* disponibles en el mercado.

Arena (de Rockwell Automation):

- Aunque carece de herramientas para implementar simuladores basados en web o ejecutar simulaciones distribuidas, ofrece la posibilidad de conectarse a fuentes de datos externas, lo cual pudiera brindar la oportunidad de utilizar un parseo de datos para solucionar dicho problema utilizando otras herramientas, sin embargo, dicha funcionalidad solo está disponible a partir de la licencia estándar.
- Animación avanzada a partir de la licencia básica-plus y animación 3D a partir de la licencia *Enterprise.*
- Lógica avanzada para modelación ("pull" y "kanban", entre otras) a partir de la versión estándar.

La licencia para el paquete básico cuesta alrededor de 1895 dólares, sin embargo existe licencias académicas que ofrecen la funcionalidad del paquete *Enterprise* para un determinado número de usuarios.

GPSS (de Minuteman Software):

Es una aplicación pionera en la simulación de procesos de la cual existe un contrato de licencia gratuito para uso académico, el cual, sin embargo, prohíbe la modificación del código interno del marco de trabajo y tampoco ofrece acceso en línea. A continuación se mencionan sus características principales.

- Brinda facilidades para el análisis de datos de salida.
- Contiene un lenguaje de simulación integrado para insertar lógica en cualquier parte del modelo.
- Contiene bloques para implementar funciones pre-programadas.
- La representación del progreso de la simulación es algo primitiva en comparación con otras herramientas.

Extend (de Imagine That Inc.):

En sus versiones más completas presenta grandes capacidades para mejorar el acceso a la aplicación (existe la opción de utilizarlo en internet, pero no es utilizable en modo multiusuario y se prohíbe el acceso en línea) y la generación de interfaces y análisis. Sin embargo, debido a los costos de licencia se describen únicamente atributos de las versiones básica y para investigación de operaciones, para las cuales se pueden negociar costos de licencia para usuarios concurrentes (múltiples) y para uso didáctico, que son aproximadamente de 995 y 1795 dólares respectivamente.

La versión para investigación de operaciones cuenta con las siguientes características:

- Herramientas de análisis estadístico con intervalos de confianza en el caso de la versión para investigación de operaciones.
- Base de datos interna.
- Alarmas y ventanas para realizar ajustes de parámetros durante la ejecución.
- Ambiente para crear interfaces personalizadas
- Optimizador, para determinar los parámetros óptimos para el modelo.
- Conexión de parámetros internos con fuentes externas de información.
- Habilidad para crear modelos jerárquicos (con sub-modelos)

2.5.2. Análisis Crítico de las Herramientas de Simulación para el Aprendizaje de Ingeniería Industrial

Los simuladores de tipo *"pick and click"* son generalmente utilizados para modelar grandes segmentos de fábricas de acuerdo a un conjunto determinado de relaciones lógicas y propiedades promedio del sistema, como pueden ser los tiempos promedio de los procesos. Sin embargo, dichas herramientas generalmente resultan limitadas en situaciones más complejas, en las que se necesite considerar mezclas de productos diferentes en los lotes de salida, con su consecuente impacto en los tiempos del sistema de producción y los inventarios en proceso, cuando se requiera modelar paradigmas de producción como el de *push* o el de *pull*, y para modelar aspectos complejos comunes en los sistemas de manufactura flexible como es el efecto en los tiempos de los mecanismos de control de colisiones, los constantes cambios de herramienta, los tiempos de los procesos y la misma secuencia de estos últimos, así como los mecanismos de coordinación entre las diferentes máquinas que compongan el sistema en cuestión. Las limitaciones anteriormente mencionadas hacen que en ocasiones sea necesario utilizar simuladores de máquinas individuales, o pequeños grupos de estos, en combinación con simuladores de tipo comercial para la simulación de una planta entera. [Fuller, 1989]

A pesar de que en la práctica la mayoría de los escenarios requieren de modelaciones y de modificaciones estándar y, por lo tanto, pueden ser modelados adecuadamente por las herramientas de tipo *pick and click*, en algunas ocasiones es necesario crear representaciones más amplias y detalladas de ellos [Schulze, 2000], además, de acuerdo con [L'Ecuyer, 2005], las herramientas gráficas y automáticas de las aplicaciones mencionadas tienden a disminuir la velocidad de ejecución de las simulaciones drásticamente, lo cual es especialmente indeseable para situaciones en las que se requiera procesar de grandes cantidades de datos y para aplicaciones en línea, es aquí donde los lenguajes especializados cobran importancia, dichos lenguajes facilitan la creación de simuladores al proporcionar herramientas para implementar elementos comunes de simulación.

En conclusión, los ambientes gráficos comerciales de simulación dan facilidades limitadas para introducir lógica de cierto grado de complejidad en las simulaciones, de manera que resultaría difícil introducir las reglas y secuencias mediante las cuales funciona la celda de ensamble, además de las limitaciones que habría para integrar diferentes herramientas de apoyo al sistema. En general se contaría con poca flexibilidad para el desarrollo de la parte funcional del simulador.

Por otro lado, el que los ambientes gráficos de programación sean lentos y poco eficientes computacionalmente, los hace imprácticos si la simulación va a ser ejecutada en dispositivos móviles, que disponen de capacidades limitadas. Inclusive si se utilizara una arquitectura cliente-servidor, el contemplar el uso del simulador por parte de una cantidad considerable de usuarios hace de dicho tipo de ambientes una herramienta poco atractiva. De modo que el desarrollo por medio de dichas herramientas se ha descartado.

2.6. Fundamentos para el Desarrollo de Simuladores

En esta sección se describen las herramientas, arquitecturas y formalismos disponibles para el desarrollo e implementación de simuladores distribuidos. Primero se habla del desarrollo de la funcionalidad de los simuladores (la parte de la aplicación encargada de realizar las simulaciones), posteriormente de los marcos de trabajo disponibles para la implementación en código y, finalmente de las arquitecturas y marcos de trabajo para el desarrollo de simuladores distribuidos.

2.6.1. Fundamentos para el Desarrollo y la Implementación de la Lógica de Simulación

Esta sección se habla de la modelación de sistemas dinámicos, los diferentes formalismos para describirlos y su implementación en código. Dicho de otro modo, esta sección describe las técnicas que pueden ser utilizadas para el desarrollo de la funcionalidad del simulador.

2.6.1.1. Modelación de Sistemas Dinámicos Discretos

Kiviat define a la simulación dinámica como: "el uso de un modelo numérico para estudiar el comportamiento de un sistema a amera que este opera a través del tiempo"[6]. Banks y colaboradores definen a la simulación dinámica discreta como: "La modelación de sistemas en los cuales la variable de estado cambia solamente en un conjunto discreto de puntos en el tiempo"[7], definiendo Sistema como: "una colección de entidades que interactúan entre ellas a través del tiempo para lograr una o varias metas"[8] y estado del sistema como: "un conjunto de variables que contienen toda la información necesaria para describir el sistema en cualquier momento"[7].

De acuerdo con [Kiviat, 1967] la estructuración de un modelo de simulación va a depender del tipo de problema que se pretende resolver, así como del sustento teórico-técnico que lleva detrás, los aspectos de interés del sistema del mundo real que se pretende modelar y la precisión requerida. A su vez, el tipo de herramientas a utilizar para crear un simulador van a depender igualmente de estas mismas circunstancias.

De acuerdo con [Kivat, 1967] los sistemas se distinguen entre sí por sus estructuras estática y dinámica. La estructura estática se refiere a todas las entidades del sistema, elementos de interés del sistema de acuerdo con [Balci, 1988], como pueden ser máquinas, personal, artículos, entre otros), sus atributos (características) y relaciones de pertenencia (conexiones entre entidades, como pertenecer a una familia o agrupación). Por el otro lado, la estructura dinámica se refiere a las actividades en las que se involucran las entidades de del sistema. [Balci, 1988] define actividad como aquello que cambia el estado de un objeto a través de tiempo y que es a su vez activado y desactivado por eventos, que son definidos por Banks y colaboradores como "una ocurrencia instantánea que puede cambiar el estado de un sistema"[9].

---

[6] Kiviat, P.J. 1967. *Digital Computer Simulation: Modeling Concepts*, Rand Corp, SantaMonica Ca, p. 17, consultado en Agosto de 2009, <http://www.randproject.org/pubs/research_memoranda/2006/RM5378.pdf>

[7] Banks, J, Carson, J, Nelson, BL & Nicol, DM 2001, *Discrete-Event System Simulation*. Prentice Hall, 2001, p. 14.

[8] Idem, p. 9.

[9] Idem, p. 10.

De acuerdo con [Jacobs, 2004], la estructura dinámica de los sistemas puede ser descrita de acuerdo a diferentes puntos de vista o formalismos, de los cuales distinguen tres formalismos clásicos para la modelación de sistemas dinámicos discretos:

- **Programación de Eventos.**- También conocida como orientación a eventos. Cosiste en generar una lista de eventos ordenada en función de su tiempo de ocurrencia, para que se obtenga de esta lista iterativamente el evento con el tiempo inmediato de ocurrencia y se invoque el método especificado en dicho evento (o, dicho de otro modo, que se generen los efectos del evento en cuestión en el sistema).
- **Escaneo de actividades.**- Consiste en definir un conjunto de eventos contingentes que sean activados por algún estado determinado del sistema. Al tener que darse un continuo monitoreo de las condiciones relacionadas a dichos eventos, generalmente resulta en simulaciones con una baja eficiencia en ejecución computacional en comparación con la programación de eventos.
- **Interacción de procesos.**- También conocida como orientación a procesos. Consiste en que cada proceso en el modelo de simulación describa su propia secuencia de acciones que conforman su ciclo de vida (como pueden ser "activar proceso", "liberar artículo", "suspender proceso", entre otros), estas secuencias de acciones serían ejecutadas de manera concurrente, con la posibilidad de que los distintos procesos se puedan comunicar e inclusive activar y desactivar acciones los unos a los otros.

En cuanto a código, Cada uno de los formalismos presenta ciertas ventajas y desventajas, según [L'Ecuyer, 2005] la interacción de procesos permite la generación de código más compacto en comparación con la orientación a eventos debido a su manera natural de describir sistemas complejos. En una simulación orientada a objetos, se considera que el sistema modelado se compone de objetos que interactúan entre sí a medida que avanza la simulación, y sus acciones y estados son representados por métodos y datos respectivamente, por lo que la implementación al usar ambos paradigmas resulta muy intuitiva (modelando los procesos como objetos), sin embargo, su ejecución es considerablemente más lenta en comparación con la más tradicional orientación a eventos, y, de acuerdo con [Jackobs, 2004], ya que dicho formalismo

generalmente es implementado usando múltiples hilos de ejecución, asignando cada uno de estos hilos a cada proceso modelado, se hace un uso considerable de recursos de CPU y de memoria, empleados para generar las instancias de dichos hilos, los cuales además llevan generalmente un registro de archivos abiertos, así como de permisos de archivos y usuarios.

Lo anterior trae como consecuencia mayores tiempos de ejecución, especialmente cuando existen grandes cantidades de procesos simultáneos en el sistema, más aún si se utilizan hilos de ejecución reales, como es el caso de Java, lo cual se puede observar en las pruebas realizadas por [Tiacci, 2007], [L'Ecuyer, 2005], y [Codl, 2003], en dichas pruebas se puede observar además cómo los tiempos de ejecución requeridos cuando se utilizan varios hilos de ejecución aumentan a medida que más elementos de la simulación son modelados como objetos de tipo proceso y ,en general, a medida que aumenta el trabajo en proceso (WIP por sus signasen inglés), que determina la cantidad de procesos en paralelo llevados a cabo.

Además, en [Jackobs, 2004] se señala que debe tenerse en cuenta que al utilizar el paradigma orientado a procesos se pueden ver afectadas la reproducibilidad de los experimentos y su precisión debido a que la ejecución y administración de hilos de ejecución concurrentes varia de un sistema operativo a otro. También se debe tener en cuenta que el uso de los métodos Thread.stop, Thread.suspend y Thread.resume, que son generalmente usados para este tipo de simulaciones y que son provistos por los mismos hilos de ejecución del sistema operativo, lleva a la generación de código propenso a quedarse estancado en puntos muertos o "deadlocks".

2.7. Programación de Simuladores en Java

De acuerdo con [L'Ecuyer, 2005] los lenguajes especializados de simulación resuelven las limitaciones de los ambientes de simulación gráficos relacionados a la falta de flexibilidad que se le brinda al usuario para representar sistemas menos convencionales que aquellos para los que las herramientas *pick and click* son creadas, así como los relativos a la baja velocidad de ejecución de dichos ambientes. Sin embargo consideran que los compiladores y herramientas

para los lenguajes especializados cuestan más y son menos disponibles en comparación con los lenguajes de uso general y enlistan, además, las siguientes ventajas de los simuladores en Java:

- Herramientas, librerías, optimizadores en tiempo de ejecución e interfaces con otro software extensivo, y de gran calidad para el desarrollo en Java.
- Funcionan en cualquier computadora sin ningún cambio.

De hecho [L'Ecuyer, 2002] consideran a los lenguajes especializados para simulación como limitados para modelar sistemas en cierto grado complejos en comparación con los lenguajes de uso general, agregando además el tiempo para aprender lenguajes especializados como una razón más para no optar por dicha opción, al considerar su aprendizaje como "una inversión de tiempo no-negligible"[10].

[Healy, 1997] por su parte agrega las siguientes razones para optar por Java en lugar de un lenguaje especializado de simulación:

- Cuenta con un marco de trabajo que facilita la implementación de un diseño orientado a objetos y sus capacidades para crear programas flexibles, reusables y nodulares.
- El hecho de que Java incluya soporte nativo en aspectos como funcionamiento en red, protocolos de Internet comunes, conexiones a bases de datos, uso de objetos distribuidos e interfaces gráficas de usuario.
- El rápido aumento de recursos para el apoyo hacia Java como lenguaje de programación.

Es importante mencionar que el uso de la máquina virtual de Java (JVM) no solamente brinda portabilidad a las aplicaciones escritas en Java sino que también brinda seguridad al sistema, ya que todo el daño que se pueda generar al ejecutar este tipo de programas queda restringido a la máquina virtual de Java. [Knudsen, 2008]

[10] L'Ecuyer, P, Meliani, L &Vaucher, J 2002, SSJ: 'A Framework for Stochastic Simulation in Java'. *Proceedings of the 34th Conference on Winter Simulation*, Winter Simulation Conference, pp. 234, consultado en Abril de 2009 en ACM Digital Library.

Existen diversos marcos de trabajo que ayudan para el desarrollo de simuladores en Java, [L'Ecuyer, 2005] mencionan los siguientes: Silk, Java Simulation, J-Sim, J-SIM, Simikit, DSOL y SSJ. SSJ, por ejemplo, provee funciones de distribución de probabilidad y pruebas de bondad de ajuste generadores de números aleatorios, colectores de estadísticas, herramientas para la administración de listas de eventos y facilidades de alto nivel orientadas hacia la simulación de procesos.

Existen paquetes tanto para desarrollo de simuladores basados en eventos (e.g. SSJ) como para el desarrollo de simuladores basado en procesos (e.g. Silk y DSOL, este último convierte simulaciones basadas en procesos a simulaciones basadas en eventos en tiempo de ejecución). Tanto [L'Ecuyer, 2005] como [Jcobs, 2004] mencionan que comúnmente, para implementar en Java un simulador basado en procesos generalmente se asigna a los objetos de tipo Proceso, los cuales van a modelar procesos independientes del sistema, un hilo de ejecución. De acuerdo con [L'Ecuyer, 2005] esto se debe evitar, excepto con el uso de la versión 1.3.1 del paquete de desarrollo de java, ya que esta es la única que soporta la creación de paralelismo simulado y el resto utiliza hilos de ejecución reales del sistema operativo lo cual no brinda un uso seguro, produce una sobrecarga de trabajo de CPU y limita la cantidad de elementos que pueden ser modelados como procesos. [Jacobs, 2004] por su parte mencionan que la implementación de simuladores basados en procesos que usa multihilos se apoya en el uso de métodos pertenecientes a los hilos de ejecución Thread.stop, Thread.suspend y Thread.resume para modelar estados del proceso en cuestión, y el uso de dichos métodos lleva al desarrollo de código proclive a la generación de puntos muertos (*deadlocks*). Además, mencionan que, al ser dependiente el manejo de múltiples hilos de ejecución del sistema operativo utilizado, el simulador desarrollado sería dependiente tanto de la plataforma como del contexto en que es utilizado.

[Jackobs, 2004] proponen el uso de un intérprete que transforma código orientado a procesos a código orientado a eventos en tiempo de ejecución, de manera que se utilicen los beneficios del

formalismo orientado a procesos (la generación de código más intuitivo y compacto) pero se eviten los problemas que conlleva, dicho interprete es usado en el marco de trabajo DSOL. Sin embargo, de acuerdo con los mismos [L'Ecuyer, 2005], ambas implementaciones mencionados del uso del formalismo basado en procesos requieren de un considerable mayor tiempo de ejecución en comparación con la implementación del formalismo orientado a eventos, especialmente cuando existen grandes cantidades de procesos simultáneos en el sistema y más aún si se utilizan hilos de ejecución.

2.7.1. Arquitecturas y Herramientas para el Desarrollo de Simuladores Distribuidos

En esta sección se describen marcos de trabajo y arquitecturas que pueden ser utilizados para el desarrollo de simuladores, útiles para desarrollar simuladores distribuidos, accesibles a través de Internet y generar código con un bajo nivel de acoplamiento. Se describen en particular la Arquitectura de Alto Nivel, el uso de componentes de software para el desarrollo de simuladores y la arquitectura basada en servicios.

2.7.1.1. Arquitectura de Alto nivel

De acuerdo con [McLean, 2007], la arquitectura de alto nivel (HLA por sus siglas en inglés), es un estándar para integrar diferentes programas de simulación, ya sea en un mismo equipo o en varios distintos, el cual funciona integrando los diversos simuladores individuales, llamados federados, en un marco llamado federación. Dicho marco o federación se compone de tres partes:

1. Un conjunto de reglas que los simuladores federados deben seguir
2. infraestructura en tiempo de ejecución (RTI por sus siglas en inglés) que define una interfaz que provee servicios para la comunicación entre los componentes federados.
3. El templete de modelo de objeto (OMT por sus siglas en inglés), el cual permite describir la información a ser intercambiada entre los componentes federados.

Para mantener funcionando dicha aplicación distribuida es necesario cuidar que los datos manejados a lo largo de ella signan las mismas definiciones y se evite la duplicidad de estos mismos, para ello es necesario crear un modelo de datos adecuado y un elemento de la aplicación

llamado *organizations*, el cual mantiene un registro de las relaciones entre los diferentes componentes de la aplicación y la información que estos requieren intercambiar. [McLean, 2007]

2.7.1.2. Simulación Basada en Componentes

Hay autores que proponen el uso de componentes de software para el desarrollo de simuladores, como es el caso de [Verbaek, 2004], quien utiliza el término "bloques de construcción" (*building blocks*) para referirse a la aplicación de componentes de software a las disciplinas de modelación y simulación, definiendo el desarrollo de software basado en componentes como "El uso de componentes reusables con una interfaz claramente definida"[11] y a los bloques (componentes) como "unidades auto-contenidas, reutilizables y reemplazables, que encapsulan su estructura interna y proveen servicios útiles o funcionales a través de interfaces precisamente definidas"[11]. De manera que los bloques de construcción pueden ser automatizados para encajar con los requerimientos específicos del entorno en que sea conectado o utilizado. De este modo, los aspectos más importantes para el desarrollo basado en componentes son: las interfaces, utilizadas como medios para caracterizar componentes, y la arquitectura (estructura que provee el contexto de integración de los componentes).

Los componentes presentan ventajas tanto para el desarrollo de aplicaciones distribuidas como para la generación en general de aplicaciones más fáciles de mantener y adaptar, ya que, de acuerdo con [Verbraeck, 2004], los bloques pueden ser construidos independientemente y mediante el uso de herramientas diferentes siempre y cuando cumplan con la especificación preestablecida para las interfaces. De modo que para modificar un bloque componente es únicamente necesario cambiar su estructura interna, sin necesidad de cambiar la interfaz con el resto del sistema, siendo posible además crear una librería de diferentes componentes que utilicen la misma interfaz para probar diferentes configuraciones de los componentes del simulador, e inclusive existe la posibilidad de programar sistemas de control que implementen las interfaces preestablecidas, tanto para establecer comunicación con otros componentes del

[11] Verbraek, A 2004, 'Component-based Simulations: the Way Forward?', *Proceedings of the eighteenth workshop on parallel and distributed simulation*, p. 142, consultado en Abril de 2009 en IEEE Xplore Digital Library.

sistema como para comunicarse con elementos del simulador con motivo de realizar pruebas de diferentes escenarios virtuales.

Un ejemplo del uso de componentes es representado por la Arquitectura de Componentes Autónomos[12] (ACA por sus siglas en inglés), el cual consta de componentes de software que, en lugar de interactuar directamente con el resto de los componentes, se comunican únicamente con sus puertos, que son unidades de software que se encargan de manejar la información entre el componente y el mundo exterior, existiendo 2 tipos de contratos: los contratos de los puertos, que definen la forma en la que los puertos van a intercambiar información con los componentes conectados a ellos y los contratos de los componentes, que únicamente son especificaciones de "caja negra para las funciones requeridas para el módulo en cuestión"[12].

2.7.1.3. Arquitectura basada en Servicios

De acuerdo con [Singh, 2004], los servicios web (*web services*) permiten implementar una arquitectura basada en servicios (SOA por sus siglas en inglés). Este tipo de arquitectura promueve la reutilización de servicios, que son componentes más amplios que los componentes usados en la arquitectura basada en componentes, al agruparse en relación a un servicio, de acurdo con Mahmud, los servicios son "la implementación de funcionalidades de negocios bien definidas"[13]. Según [Singh, 2004], los servicios se centran en la funcionalidad provista por sus interfaces (se comunican entre ellos y con los clientes por medio de interfaces bien definidas y conocidas). Este tipo de Arquitectura permite al cliente invocar servicios a través de las interfaces que estos proveen desde la red.

El *World Wide Web Consortium* (W3C) define a los servicios web de la siguiente manera: "Un servicio Web es un sistema de software identificado por un URI cuyas interfaces públicas y enlaces están definidos y descritos usando XML. Su definición puede ser descubierta por otros

[12] j-sim official 2003, *Tutorial: Working with J-Sim*, <http://sites.google.com/site/jsimofficial/j-sim-tutorial>
[13] Mahmoud, QH 2005(b), *Service-Oriented Architecture (SOA) and Web Services: The road to Enterprise Application Integration (EIA)*, consultado en Mayo de 2009, <http://www.oracle.com/technetwork/articles/javase/soa-142870.html>

sistemas de software. Estos sistemas pueden interactuar entonces con el servicio Web de la manera prescrita en su definición, usando mensajes basados en XML transportados por protocolos de internet"[14].

De acuerdo con [Singh, 2004] en una arquitectura basada en servicios se compone de lo siguiente:

- Un servicio que implementa la lógica de negocio y la expone a través de interfaces bien definidas.
- Un registro donde el servicio publica sus interfaces para permitir al cliente descubrir el servicio.
- Clientes (que pueden a su vez ser servicios) que descubren el servicio usando los registros y acceden al él directamente a través de interfaces expuestas.

2.7.2. Análisis Crítico de los Fundamentos para el Desarrollo de Simuladores

En esta sección se hace un análisis de los fundamentos para el desarrollo de simuladores presentados en este documento, desde el desarrollo de la lógica funcional, que es la parte de la aplicación que va a simular el funcionamiento del sistema a analizar, hasta las herramientas de programación, los marcos de trabajo y las arquitecturas que se pueden utilizar.

2.7.2.1. Análisis Crítico de los Formalismos y Herramientas para el Desarrollo de Simuladores

A continuación se presenta un análisis de los formalismos y herramientas que pueden ser utilizados en el desarrollo de la funcionalidad del simulador.

[14] W3C 2004, *Web Services Architecture* <http://www.w3.org/TR/ws-arch/#whatis>

Formalismos Basados en Eventos y Procesos

Si bien el formalismo orientado a procesos representa grandes ventajas en lo que se refiere a la generación de código compacto, fácilmente entendible y por lo tanto de fácil mantenimiento o actualización, cuando se contempla el uso del simulador por medio de dispositivos móviles y de forma distribuida resulta inconveniente usar dicho formalismo, ya que eso implicaría una implementación basada en múltiples hilos de ejecución, siendo que los hilos de ejecución utilizan una cantidad considerable de memoria y los dispositivos móviles tienen recursos limitados de ese tipo. Por otro lado, el implementar una aplicación de forma distribuida requiere mantener una comunicación constante entre los diferentes módulos de la aplicación, lo que muy generalmente implicaría el uso de hilos de ejecución adicionales para implementar clases de tipo *listener* que se mantengan a la espera de respuestas de los módulos con los cuales se va comunicar cada módulo del sistema, lo que impactaría el desempeño general de la aplicación.

El formalismo basado en eventos presenta ventajas en lo que se refiere al uso de memoria y tiempos de ejecución, sin embargo, podría requerir de ciertas modificaciones para desarrollar una aplicación que pueda funcionar de forma distribuida, para evitar la necesidad de que los módulos se tengan que coordinar constantemente con un programador de eventos externo.

En conclusión, dado que el simulador va a funcionar en dispositivos móviles, el paradigma orientado a eventos resulta más atractivo, sin embargo, para desarrollar una aplicación cuya funcionalidad es distribuida y para generar código más fácil de desarrollar y mantener el paradigma orientado a eventos resulta ventajoso. De hecho, el utilizar el paradigma orientado a eventos implicaría realizar modificaciones sobre dicho paradigma al implementar la funcionalidad de manera distribuida, ya que, debido a su naturaleza, el código de sus diferentes componentes quedaría muy acoplado. Las observaciones realizadas en esta sección serán utilizadas para la elección del paradigma de modelación en el capítulo 3.

Herramientas para el Desarrollo de Simuladores en Java

Si bien existen marcos de trabajo para el desarrollo de simulaciones en java basados en eventos, los cuales pueden agilizar el desarrollo al brindar herramientas desarrolladas por gente con experiencia en el desarrollo de este tipo de aplicaciones, al considerar la necesidad de implementar el simulador para brindar acceso desde dispositivos móviles, así como la posibilidad de ejecutar las simulaciones de forma distribuida, existiría la necesidad no solo de aprender a utilizar el marco de trabajo elegido sino también el código del marco de trabajo de manera detallada para realizar modificaciones a este.

Los dispositivos móviles tienen una capacidad de memoria considerablemente menor que la de las computadoras portátiles y de escritorio actuales, razón por la que utilizan una especificación especial de la máquina virtual de java y cuentan con una librería de aplicaciones más reducida que incluye menos clases y métodos en su paquete de desarrollo (*kit* de desarrollo), de manera que posteriormente pudiera ser necesario hacer modificaciones para evitar el uso de funciones no disponibles en la Interfaz de Programación de Aplicaciones (API por sus siglas en inglés) para dispositivos móviles. Adicionalmente sería muy probable que existiese la necesidad de intentar modificar el código para ajustar el uso de recursos computacionales a las capacidades de los dispositivos móviles.

Por el otro lado, el implementar la aplicación de forma distribuida implicaría realizar modificaciones en el código para dar acceso a aplicaciones remotas a diferentes funciones de tipo *getters* (usados para permitir que alguna variable de un objeto sea utilizada sin tener que hacerla directamente accesible desde afuera de dicho objeto) *y setters* (usados para permitir que alguna variable de un objeto se modificada sin tener que hacerla directamente accesible desde afuera de dicho objeto). En el caso de algunas herramientas, como es el de *Java Simulation*, existen herramientas para la implementación de simuladores distribuidos. *Java Simulation* se basa en el

uso de componentes por medio de un script que describe la interacción entre los diferentes componentes que conforman la aplicación.

Considerando los eventuales problemas que pudieran darse al hacer la aplicación disponible para dispositivos móviles, la opción de desarrollar el simulador sin hacer uso de ningún marco de trabajo para simulación, y por lo tanto teniendo un completo conocimiento de todo código y lógica funcional utilizados representa una buena alternativa, especialmente considerando que el sistema a modelar no es exageradamente complejo.

2.7.2.2. Análisis Crítico de las Arquitecturas y Herramientas para la Creación de Simuladores Distribuidos

Hacer uso de herramientas disponibles en las librerías de aplicaciones disponibles en el kit de desarrollo de java que manejen los flujos de datos (*streams*) y la serialización de objetos sería esencial para el desarrollo de una aplicación distribuida, un ejemplo de dichas herramientas es de Invocación Métodos Remotos (RMI por sus siglas en inglés), el cual es útil para el desarrollo de aplicaciones distribuidas en Java. Por otro lado, el uso de una arquitectura basada en componentes podría resultar ser una muy buena opción para generar una aplicación ampliamente reusable y fácil de mantener, también existe la opción de utilizar de forma moderada algún marco de trabajo que proporcione facilidades para implementar dicha arquitectura, sin embargo, la efectividad de esta solución será limitada por la existencia de herramientas disponibles para dispositivos móviles.

Sin embargo, dado lo lentas y poco confiables que son las redes inalámbricas, así como la naturaleza simple de los diferentes módulos del simulador y las capacidades limitadas de los dispositivos a utilizar, el implementar la ejecución del simulador entre diferentes máquinas no parece ser conveniente. Resultaría más rápido y eficiente ejecutar completamente las simulaciones en una sola computadora que funcione como servidor.

Se puede logara un nivel de portabilidad considerable al implementar la parte funcional del simulador como servicio Web ya que de esta manera, la lógica funcional queda como un componente independiente que se comunica con las aplicaciones cliente, las cuales por su parte se encargarían de realizar la presentación y la interfaz.

El desarrollar la aplicación en base al patrón Modelo-Vista-Controlador (MVC) representa una buena opción, al separar dicho trabajo en sus principales aspectos (modelo o funcionalidad, vista o interfaz de usuario, y el control general de la aplicación). Por otro lado, el utilizar el servidor para realizar toda la lógica funcional del simulador, liberaría al dispositivo del usuario, que pudiera ser un dispositivo móvil, de esa responsabilidad, de manera que este último podría ahorrar recursos de memoria y procesamiento, los cuales, en el caso particular de los dispositivos móviles son escasos.

De este modo el dispositivo móvil podría utilizar mayor cantidad de recursos para la interfaz de usuario, de este modo facilitando eventualmente la creación de una presentación gráfica más elaborada, y la lógica funcional no se vería limitada por las capacidades del dispositivo móvil del usuario. Además, esta arquitectura, agiliza el portar aplicaciones hechas en la versión estándar de Java a dispositivos móviles, al no necesitar ningún tipo de modificación en su código para ajustarlo a las APIs comprendidas por los estándares seguidos para el desarrollo de aplicaciones móviles, principalmente el Perfil de Dispositivo de Información Móvil (MIDP por sus siglas en inglés), y a las capacidades de la versión de Máquina Virtual de Java que utiliza dicho tipo de dispositivos móviles. Además, al seguir este patrón se pueden crear diferentes interfaces de usuario sin realizar cambios en la lógica funcional.

La arquitectura basada en servicios Web (sección 7.2.1.3) podría ser usada para el desarrollo de la aplicación, para, de este modo, separarla en componentes independientes, de manera que el desarrollo de la interfaz de usuario pueda ser desarrollada en cualquier lenguaje, plataforma y por cualquier autor que, aunque no esté familiarizado con el desarrollo del simulador, solamente

tenga que implementar un cliente del web servicio Web que haga uso de los servicios de este. Esto además permitirá combinar el servicio en cuestión con otros complementarios que podrían consistir en el uso de sistemas asistentes inteligentes o aplicaciones para determinar programas de producción óptimos, entre otros.

La arquitectura basada en servicios Web utiliza grupos más amplios de software que los utilizados en la arquitectura basada en componentes, al enfocarse al servicio que van a ofrecer. El utilizar servicios Web para implementarla permite distribuir la aplicación a través de la red entre diferentes plataformas y lenguajes de desarrollo. Además, el servicio Web se utilizaría como controlador para implementar a su vez el patrón MVC, para que el desarrollo y el mantenimiento de la lógica funcional, el controlador y la vista fueran en cierto grado independientes, y desarrollados por separado.

Por otro lado, aunque el simulador (i.e. la lógica funcional) puede ser implementado separando los diferentes módulos en componentes, al no tener estos un tamaño considerable y al no estar el simulador completo desarrollado por diferentes personas (i.e. con diferentes herramientas o de forma independiente), no parece ofrecer ventajas considerables.

Los puntos mencionados en esta sección servirán para que se determine qué herramientas y que técnicas se utilizarán para el desarrollo de la aplicación como sistema distribuido accesible desde dispositivos móviles. De los mismos puntos se puede descartar el desarrollo de un simulador basado en componentes.

2.8. Desarrollo de Aplicaciones Móviles

De acuerdo con [Wains, 2008] con el aumento en las capacidades de computación de los dispositivos móviles, que iguala las capacidades que alguna vez tuvieron las computadoras, junto con la penetración que dichos dispositivos tienen alrededor del mundo hacen de esta tecnología una importante opción para dar material educativo a distancia a los alumnos. Además,

el uso de dispositivos móviles brinda la oportunidad de brindar mayor flexibilidad al estudiante para realizar sus actividades académicas, pudiendo éste interactuar con sus compañeros y acceder a material educativo "eliminando las limitaciones de tiempo y espacio en el aprendizaje, permitiendo a los alumnos aprender en donde y cuando sea"[15].

2.8.1. Sistemas Operativos para Dispositivos Móviles

Actualmente existe una cantidad considerable de dispositivos y sistemas operativos en el mercado, por lo cual, ha habido diversos esfuerzos por crear compatibilidad entre los diferentes sistemas operativos usados en los diferentes dispositivos existentes. Uno de los primeros esfuerzos en la industria fue la creación del sistema operativo Symbian, desarrollado por un conglomerado compuesto por Nokia, Erickson, Panasonic y Samsung. Para este sistema operativo posteriormente surgieron cuatro diferentes interfaces: Crystalm, Sapphire, Perl y Quartz. Dichas interfaces estaban dirigidas a diferentes dispositivos de hardware y compañías en específico. Ericson, Sony, Panasonic y Samsung terminaron por retirarse del esfuerzo, dejando a Nokia como único propietario [Hall, 2009]. Actualmente Symbian soporta MIDP, que es un estándar para aplicaciones en Java, además de ser *Open Source* y gratuito.

De acuerdo con [Hall, 2009], en los últimos años, la carrera entre los sistemas operativos se ha perfilado a convertirse en una competencia entre Android[16], desarrollado por Google, e iPhone[17], desarrollado por Apple. Entre las ventajas que presenta Android se encuentra la posibilidad de desplegar páginas web normales, sin necesidad de acceder a páginas creadas específicamente para dispositivos con pantallas pequeñas (como puede ser en wml). Además, Android es un sistema operativo abierto basado en Linux, y los desarrolladores pueden crear aplicaciones en java desde cualquier sistema operativo para ser ejecutadas en él. Esto además de varias otras ventajas, como el hecho de que están basadas en redes 3G (considerablemente más rápidas que los esquemas existentes de redes para dispositivos móviles, aunque aún limitado a 1mb/s como velocidad promedio de descarga), y que fue creado para tomar ventaja de los "*touch screen*". De

[15] Sheng, H, Siau, K, Fui-Hoon Nah, F 2010, *Understanding the values of mobile technology in education: a value-focused thinking approach*, ACM SIGMIS Database, p. 26, consultado en Marzo de 2010 en ACM Digital Library.
[16] <http://www.androidgoogle.net/>
[17] <http://www.apple.com/mx/iphone/>

acuerdo con los mismos autores, Android es un verdadero sistema operativo y sus aplicaciones son exclusivamente desarrolladas en Java.

iPhone por su lado, a pesar de poseer una ventaja considerable en el mercado de dispositivos móviles (se encuentra en tercer lugar de participación de mercado, solo detrás de Symbian y RIM[18]), posee considerables desventajas, que de acuerdo con [Hall, 2009] radican principalmente en las facilidades de desarrollo, ya que iPhone requiere del uso de un ambiente de desarrollo propietario y cerrado, que, a pesar de que se puede descargar gratis, sólo corre en sistemas operativos Mac. De manera que desarrollar aplicaciones para iPhone pudiera no ser del todo atractivo para los millones de desarrolladores de Java que utilizan PC, al tener que familiarizarse estos con el ambiente de desarrollo para iPhone y cambiar a Mac.

Actualmente Motoraola, marca de la cual existen decenas de millones de dispositivos en el mercado, tiene planeado dividir sus dispositivos entre Android y Windows Mobile. Adicionalmente, Ericson, HTC, Samsung, entre otros productores de hardware tienen sus propios planes para lanzar al mercado dispositivos con Android. [Hall, 2009]

2.8.2. Aplicaciones Java para Dispositivos Móviles

Según [Knudsen, 2008], actualmente Java es popular entre los fabricantes de dispositivos y los operadores de redes inalámbricas (*carriers*), esto debido al uso de la máquina virtual de Java ya que esta brinda mayor seguridad a los usuarios para utilizar software de terceros, debido a que cualquier daño que pueda producir dicha aplicación queda restringido a la JVM dejando el resto del sistema intacto.

Según [Hamer, 2007], *Java Micro Edition* (JVM) es la versión de la plataforma de java que es designada para su uso en dispositivos pequeños con capacidades reducidas de procesamiento y

[18] Gartner 2010, *Gartner Says World Wide Mobile Devices Sales Grew 10.8 Percent in Second Quarter of 2010, But Competition Drove Prices Down*, revisada en Agosto de 2010, <http://www.gartner.com/it/page.jsp?id=1421013>

memoria en comparación con aquellos para los que se utiliza la edición estándar (J2SE), de acuerdo con [Nogueira, 2008] dicha versión tiene 2 clasificaciones, la Configuración de Dispositivo Conectado (CDC por sus siglas en inglés) y la configuración de Dispositivo Limitado Conectado (CLDC por sus siglas en inglés). CDC es para dispositivos fijos, como las cajas combertidoras y decodificadoras de los televisores que permiten realizar navegación web y el uso de correo electrónico sin necesidad de una computadora como tal. CLDC es para dispositivos aún más limitados en memoria y procesamiento, como los teléfonos celulares.

La configuración especifica el tipo de máquina virtual de Java que se ha de utilizar y qué se ha de incluir en las librerías de clases mínimas (en el caso de CLDC, java.* y javax.microedition.io). Sobre esta configuración se ha creado el Perfil de Dispositivos de Información Móvil (MIDP por sus siglas en ingles), el cual, de acuerdo con [Hamer, 2007] define un conjunto estándar de librerías. MIDP es una especificación creada por un consorcio de compañías e individuos a través del proceso de la comunidad de java (JCP por sus siglas en inglés), las especificaciones establecidas mediante el JCP son conocidas como Peticiones de Especificación de Java (JRSs por sus siglas en inglés). [Knudsen, 2008]

La primera especificación MIDP, MIDP 1.0, define un ambiente completo para que aplicaciones basadas en tecnología java puedan ser ejecutadas en dispositivos pequeños. MIDP 2.0, que es ampliamente usada en teléfonos celulares hoy en día, presenta nuevas características para el desarrollo de juegos e interfaces de usuarios. [Knudsen, 2008]

Además de las especificaciones de MIDP, existen otras JSR, llamadas APIs opcionales, para mensajes de texto, multimedia, bluetooth, posicionamiento global (GPS), gráficas 3D, entre otras funcionalidades. Al existir una abundante cantidad de APIs opcionales, para evitar el problema de determinar cuáles APIs están disponibles para un dispositivo determinado (en el caso de los

desarrolladores), existen las "especificaciones sombrilla"[19], que incluyen un conjunto de especificaciones y APIs opcionales. La primera de estas "especificaciones sombrilla" fue la llamada Tecnología de Java para la Industria Inalámbrica (JTWI por sus siglas en inglés), dicha especificación incorporaba MIDP 2.0, CLDC 1.1 (ó 1.0), WMA (API opcional para mensajes inalámbricos) y, de manera opcional, MMAPI (API opcional para multimedia). Además, JTWI define otros requerimientos, como mínimos de memoria, mínimos para multihilos (*multithreading*), y el comportamiento de las APIs, para de esta manera reducir la ambigüedad para los desarrolladores. [Knudsen, 2008]

Como considerablemente mayor cantidad de APIs han sido creadas desde la aparición de JTWI, surgió una "especificación sombrilla" más nueva, bajo el nombre de Arquitectura de Servicio Móvil (MSA por sus siglas en inglés). Esta última especificación se clasifica en dos subconjuntos: MSA y MSA-subconjunto, agrupando el subconjunto APIS para dispositivos que no cuentan con las capacidades de hardware suficientes como para soportar el conjunto completo de MSA. Las aAPIS que únicamente pertenecen al conjunto MSA y no se incluyen en el subconjunto son llamadas "APIs condicionalmente obligatorias"[20], mientras que las que se encuentran incluidas en ambas son llamadas "APIs obligatorias"[20]. [Knudsen, 2008, p.7]

De acuerdo con [Hamer, 2007], las principales diferencias entre la JVM estándar y la JVM especificada en CLDC se encuentran en modificaciones relativas al colector de basura de java (*garbage collector*), la seguridad, el manejo de hilos (*threads*) y, el manejo de excepciones y errores. Otras diferencias radican en la fase de pre-verificación después de la compilación en la versión de CLDC y en que las plataformas que cumplen con CLDC únicamente permiten el uso de caracteres comprendidos por el rango latín 1 del estándar Unicode versión 3.

---

[19] Knudsen, Jonathan 2008, *Kicking Butt with MIDP and MSA: Creating Great Mobile Applications*, Prentice Hall, EUA, p.5

[20] Idem, p.7.

Existen además modificaciones dentro de las librerías que han sido reducidas y modificadas para usar clases más eficientes, de acuerdo con [Nogueira, 2008], los principales retos al migrar aplicaciones que cumplen con J2SE a MIDP radican en los cambios en las clases disponibles del marco de trabajo de colecciones, y en la eliminación de dos recursos principalmente, la clonación de objetos y el uso de reflexión.

De acuerdo con [Kenteris, 2009], la principal ventaja que ofrece el desarrollo con JME en comparación con el uso de acceso inalámbrico a Web, (principalmente por medio de WAP o i-mode) es que se le evita al usuario tener que conectarse continuamente a la red para obtener información nueva, ya que la aplicación JME puede ser descargada una sola vez con toda la información que el usuario necesite, de manera que éste se ahorre los costos económicos y los posible problemas de conexión ligados al acceso inalámbrico a Web.

Aunque existe un paquete opcional[21] para J2ME para la Invocación de métodos remotos (RMI), útil para la creación de aplicaciones distribuidas, este paquete solo se encuentra disponible para CDC, de modo que no es disponible para dispositivos móviles. Este paquete requiere la implementación completa de la máquina virtual de Java.

2.8.2.1. Desarrollo de Aplicaciones MIDP

Las aplicaciones y los diferentes recursos que éstas utilizan son empaquetados en las *midlet suites,* que, de acuerdo con [Knudsen, 2008], son archivos JAR (archivo java) que contienen las clases que componen la aplicación, un archivo *mainfest* y los diferentes recursos que utiliza dicha aplicación, como pueden ser imágenes o sonidos. Dicho JAR va acompañado de un archivo descriptor de aplicaciones Java (JAD por sus siglas en inglés), que es un archivo de texto que contiene información para ayudar al dispositivo a instalar y ejecutar la aplicación. De acuerdo con [Hemer, 2007], el archivo JAD y el archivo *mainfest* tienen mucha información en común, que de ser inconsistente se impide la instalación de la aplicación en cualquier dispositivo

[21] <http://jcp.org/aboutJava/communityprocess/final/jsr066/index.html>

MIDP por motivos de seguridad. Ambos archivos poseen información necesaria para la ejecución de la aplicación, como lo es la versión CLDC requerida. La diferencia entre los dos archivos mencionados anteriormente radica en que la información que requiere de seguridad va en el *mainfest* dentro del JAR y la información necesaria para la instalación va en el JAD, fuera del JAR. Existen, sin embargo, ambientes de desarrollo que se encargan de la creación de dichos archivos, dejando al desarrollador únicamente la tarea de desarrollar las clases.

De acuerdo con [Koller, 2009], el JAD permite al usuario obtener información de la aplicación antes de descargarla, en los casos en los que se aplica un modelo de distribución sobre el aire (*over the air*), dicho modelo permite al usuario descargar la aplicación desde un servidor a través del Browser de su dispositivo móvil.

[Camargos Tavares, 2008] proponen el uso del modelo llamado presentación remota, para ejecutar aplicaciones en dispositivos móviles, dicho modelo divide la ejecución de las aplicaciones móviles en dos partes, una llevada a cabo por el dispositivo móvil en cuestión (que consta de la interfaz de usuario) y otra llevada a cabo por un servidor (que consiste en la lógica funcional). De esta manera se puede mejorar el desempeño de aplicaciones móviles cuando estas demandan grandes cantidades de procesamiento. Dichos autores proponen así mismo un sistema llamado Presentación Remota usando Aspectos (RDA por sus siglas en inglés), dicho sistema utiliza programación orientada a eventos para adaptar aplicaciones que originalmente no utilizaban dicho modelo para introducir la interacción servidor-dispositivo móvil y bloquear la presentación original del lado del servidor.

2.8.2.2. Desarrollo de Aplicaciones MIDP para BlackBerry

RIM proporciona la posibilidad de desarrollar aplicaciones para J2ME, lo cual incluye el desarrollo de su propia máquina virtual de Java, la cual soporta la configuración CLDC y el perfil MIDP. Los dispositivos BlackBerry también incluyen un conjunto de APIs específicas que solo están disponibles para ese tipo de dispositivos, las cuales pretenden ser más sofisticadas que los MIDlet estándar y tener una integración más estrecha con los dispositivos BlackBerry. De

manera que existe la opción de desarrollar MIDlets, que pueden funcionar para cualquier dispositivo que incorpore MIDP y la opción de desarrollar RIMlets, que únicamente pueden funcionar para dispositivos BlackBerry, ya que utilizan las APIs específicas para estos dispositivos. [Mahmoud, 2005]

2.8.3. Aplicaciones Web para Redes Inalámbricas

[Savino, 2001] establece como diferencia importante entre la navegación en red por medio de dispositivos móviles y aquella por medio de dispositivos de navegación convencionales, la situación en la que usuario va a acceder a la información, ya que en lugar de "surfear la red" en busca de información el usuario querrá acceder a sitios específicos mientras se encuentra en movimiento (*on the go*), haciendo uso de servicios específicos como compras, juegos, finanzas, etc. Por otra parte, el usuario también buscaría otro tipo de servicios específicos para los dispositivos móviles, como pueden ser: de administración de llamadas, de administración de la libreta de teléfonos, etc.

2.8.3.1. Protocolo de Aplicaciones Inalámbricas

Según [Savino, 2001], el Protocolo de Aplicaciones Inalámbricas (WAP por sus siglas en inglés) fue creado por el fórum WAP, conformado por Nokia, Ericsson, Motorola y Phone.com en 1997 con el propósito inicial de definir una especificación utilizada por toda la industria para el desarrollo de aplicaciones Web para acceso a través de redes de comunicación inalámbricas. De acuerdo con [Gavalas, 2006], WAP es una aplicación de protocolos similar a la de TCP/IP. WAP, de hecho, se basa en el modelo de internet, usando tecnología de internet optimizada para hacer frente a las limitaciones de los dispositivos móviles.

El protocolo de aplicación Inalámbrica, de acuerdo con [Singh, 2009], es usado para acceder a información y servicios desde dispositivos móviles, dicho protocolo utiliza el lenguaje WML del mismo modo que HTTP utiliza HTML (esto en el caso de la versión 1.0 de WAP). Según [Savino, 2001], WML surge para afrontar la incapacidad de los dispositivos móviles para

desplegar correctamente páginas en HTML, debido principalmente al tamaño de sus pantallas. En términos generales, de acuerdo con [Singh, 2009], WAP (1.0) consiste en lo siguiente:

1. el dispositivo cliente envía una petición a la puerta de enlace (*gateway*) WAP.
2. La puerta de enlace traduce la petición a HTTP y la envía al servidor destino.
3. El servidor actúa como contenedor de las entidades del lado del servidor, como pueden ser *servlets*, *scripts* u otros.
4. El servidor transmite el resultado de la petición de regreso al WAP *Gateway*.
5. El WAP *Gateway* traduce la respuesta del servidor a WAP y la envía al dispositivo móvil que hizo la petición.

De acuerdo con [Gavalas, 2006], la versión más nueva de WAP, WAP2.0, permite utilizar el protocolo http, de manera que no sea necesaria la comunicación con un WAP *Gateway*, Además de hacer uso del lenguaje XHTML MP (perfil móvil de HTML extensible), que es un subconjunto de XHTML, para el despliegue de las páginas web, lo cual resultaría más familiar y conveniente para los desarrolladores web, y, sobre todo, mejora la portabilidad hacia la misma web. Por otro lado, la adopción de esta versión de protocolos, al basarse en el uso de paquetes hace que resulte más barato y eficiente el intercambio de información a través de la web.

2.8.3.2. i-Mode

Al igual que WAP, es un conjunto de protocolos para el acceso inalámbrico a contenido web, Este conjunto de protocolos pertenece a la empresa japonesa DoCom. Utilizó una red *packet-switched* y una versión compacta de HTML (cHTML) desde antes de que surgiera WAP 2.0. [Gavalas, 2006]

i-mode también se destacó por su modelo de negocios, que fue novedoso en su momento, al mantener a sus clientes conectados todo el tiempo y únicamente cobrarles por la cantidad de información descargada. También permite a sus usuarios navegar entre contenidos de páginas registradas con DoComo, sin la necesidad acceder a ella a través de su nombre de dominio. [Gavalas, 2006]

2.8.4. Widgets Basados en Tecnología Web

Existe por otro lado la opción de utilizar *widgets*, que son, de acuerdo con el *World Wide Web Consortium* (W3C):

"aplicaciones interactivas de un solo propósito para desplegar y/o actualizar datos en la red, empaquetados de manera que solo requieren una única descarga e instalación en la máquina o dispositivo móvil del usuario"[22].

De acuerdo con [Kaar, 2007], los *widgets* utilizan tecnología web estándar, por lo que su desarrollo es similar al de las aplicaciones web estándar. Sin embargo, dicho tipo de aplicaciones, son ejecutadas por medio de un motor de widgets (*widget engine*), y usualmente no son interoperables, funcionando únicamente en un motor widget particular. Esto debido a que los widgets son almacenados y descritos (por medio de un *mainfest*) de maneras diferentes, y debido a APIs no estándar de los mismos motores de *widgets*. Sin embargo, el *World Wide Web* consortium se encuentra actualmente trabajando en la estandarización del desarrollo de widgets. Por otro lado, también de acuedo con [Kaar, 2007] NetVibes se encuentra desarrollando un API universal de Widgets (UWA por sus siglas en inglés) que permita la ejecución de widgets en diferentes plataformas. Así mismo, de acuerdo con [Mihalic, 2008], para el año 2009, los diferentes desarrolladores de motores de widgets, han comenzado a publicar sus APIs para permitirle a los desarrolladores trabajar para lograr interoperabilidad.

Los widgets, según [Kaar, 2007], ofrecen principalmente las siguientes ventajas en comparación con los demás tipos de implementaciones nativas:

-El diseño de la interfaz de usuario es simplificada y se pueden crear fácilmente interfaces ricas.

-El proceso de desarrollo es corto.

[22] W3C 2009, *Widget Packaging and Configuration, visata en Marzo de 2010,* <http://www.w3.org/TR/2009/CR-widgets-20091201/>

-La similitud con el desarrollo web genera una importante cantidad de desarrolladores potenciales.

Por otra parte, el mismo [Kaar, 2007], reconoce que aún existen limitaciones en la portabilidad de los widgets y que estos, por su naturaleza, no son recomendables para la creación de aplicaciones con lógica compleja o videojuegos, sugiriendo el uso de widgets para aplicaciones relativamente simples y relacionadas a una actividad en concreto.

Para un adecuado desarrollo de aplicaciones web es importante separar las tareas relativas al desarrollo de software de aquellas que son relativas al diseño de las páginas, las páginas generalmente son desarrolladas por diseñadores web especializados en crear páginas de buena apariencia pero que generalmente no están familiarizados con programación en Java. Por otro lado, el utilizar gran cantidad de Java Script para generar contenido web resulta en una mala separación de los aspectos de la programación al estar fuertemente acoplados la lógica de negocios o la funcionalidad, y el diseño de la interfaz de usuario. De manera que, a pesar de que se utilizan las mismas herramientas que para el diseño Web, se requieren ciertos cambio en la perspectiva de desarrollo.

2.8.5. Análisis Crítico de las Tecnologías para el Desarrollo de Aplicaciones Móviles

En esta sección se da un análisis de las diferentes opciones, en cuanto a tecnologías disponibles, para el desarrollo de aplicaciones móviles, en relación al desarrollo de simuladores multiusuario.

Desarrollo Web para Dispositivos Móviles

El desarrollo de aplicaciones web para dispositivos móviles representa una buena opción en términos de portabilidad, ya que cualquier dispositivo que tenga un navegador para contenido en Lenguaje de Marcado Inalámbrico (WML por sus siglas en inglés) y acceso a una puerta de enlace (*gateway*) tipo WAP podrá acceder al contenido. La versión 2.0 resulta aún una mejor opción, ya que ha adquirido elementos en los que i-mode aventajaba a la versión anterior de

WAP, como el uso de un subconjunto de XHTML y el uso de redes tipo *packed-switched.* La principal diferencia sin embargo consiste en que i-mode pertenece a DoCoMo mientras que WAP es independiente de la compañía operadora.

A pesar de su capacidad de portabilidad, en el caso de simuladores en tiempo real, que requieran realizar cierta lógica, las aplicaciones Web podría no representar una opción tan buena, a pesar de que cuentan con un *script* (conjunto de instrucciones interpretadas) equivalente a JavaScript, dicho *script* no está pensado para generar lógica más allá de funcionalidad relativamente simple para mejorar el aspecto de la interfaz de usuario.

El tiempo de simulación se podría ver afectado por la velocidad de las redes inalámbricas, que son menos eficientes que las redes tradicionales, sin embargo, por otro lado, el utilizar el servidor para ejecutar simulaciones exhaustivas podría tener un impacto positivo en el desempeño de la aplicación debido a las capacidades limitadas de los dispositivos móviles.

Widgets basados en Tecnología web

A pesar de que los *widgets* basados en herramientas de desarrollo web parecen resultar una buena solución para generar mayor interoperabilidad entre las distintas plataformas *mobile*, hoy en día, al igual que los *widgets* para *desktop,* aún no son completamente portables entre los diferentes dispositivos. Aunque, por otro lado, sería de esperar que las modificaciones requeridas por dichos *widgets* fueran pequeñas, al estar basados en herramientas web estándar. Por otro lado, al realizar una aplicación con J2ME, esta sería prácticamente portable a cualquier dispositivo que incluya MIDP (esto, claro está considerando detalles como los diferentes tamaños de pantalla, capacidad de procesamiento, tipo de archivos que se usan, entre otros), sin embargo, adaptar la aplicación a plataformas que no soporten J2ME (como iPhone por ejemplo) requeriría de un mayor esfuerzo, al no hacerse uso de herramientas estándar, como css y HTML,

como en el caso del desarrollo de widgets. Es importante recalcar que la lista de dispositivos que incluyen MIDP es larga[23].

Es necesario recalcar que, en el caso de MIDP, las aplicaciones no siempre son directamente portables, por ejemplo, en el caso de RIM, aunque no es necesaria ninguna modificación del código de la aplicación, es necesario generar un archivo de tipo COD a partir de la *midlet suite* de la aplicación.

También es importante mencionar que, aunque los *widgets* basados en web pueden incluir java script, dicha herramienta tradicionalmente no es usada para generar lógica relativamente compleja, por lo que, al requerirse de una presentación detallada de la simulación y de interacción con otro tipo de aplicaciones, como por ejemplo, un tutor inteligente, pudiera, probablemente, no resultar una opción tan simple. Además, MIDP cuenta con un API de videojuegos que facilitaría y mejoraría la creación de una presentación en tiempo real de los resultados. Los Widgets por su parte, al utilizar herramientas Web estándar, no cuentan con APIs que permitan crear ágilmente animaciones o videojuegos (útiles para crear interfaces de simuladores).

El análisis comparativo presentado en esta sección será utilizado para seleccionar las herramientas para desarrollo de aplicaciones móviles a utilizar en el desarrollo del simulador móvil distribuido, considerando las limitaciones y ventajas anteriormente mencionadas. A pesar de que las aplicaciones Web y los *widgets* basados en tecnología Web aportarían una portabilidad más amplia, es importante considerar que las redes inalámbricas todavía son considerablemente lentas y poco confiable, además difícilmente se podría programar con dichas herramientas una lógica considerablemente compleja, como la del simulador y más difícilmente se podría generar una animación de las simulaciones.

[23] <http://www.club-java.com/TastePhone/J2ME/MIDP_mobile.jsp>

Capítulo 3

Modelo para Simuladores Móviles Distribuidos

Este capítulo describe la estructura, el funcionamiento, la integración y distribución de la aplicación. Se considera tanto el diseño del simulador como tal, es decir, la manera en que se desarrolla el software que va a representar la línea de ensamble que se requiere simular, así como su integración en una aplicación distribuida que pueda ser manipulada por diferentes usuarios simultáneamente a través de dispositivos móviles. De modo que el desarrollo de la aplicación se dividió en dos grandes bloques (ver Figura 3. 1):

- El desarrollo de la funcionalidad de la aplicación: que se refiere al desarrollo del simulador como tal, lo cual comprende la modelación del tipo de sistema de manera abstracta (modelación de una línea de producción cualquiera compuesta por diferentes módulos de trabajo y almacenes), la modelación del sistema de manera detallada (cómo se va a modelar cada uno de los módulos que componen el sistema a modelar en base al modelo abstracto y cómo se van a introducir sus características específicas), la programación a nivel abstracto (generación de código que represente el funcionamiento general de los diferentes tipos de componentes del sistema identificados) y la programación a detalle o implementación funcional (el uso de la programación realizada a nivel abstracto para representar en software cada uno de los componentes del sistema a modelar, la línea de ensamble en este caso).
- El desarrollo del sistema móvil distribuido: que comprendió todo lo relacionado con la estructura de la aplicación, la manera en que el usuario va a acceder a la aplicación y la manera en que va a interactuar con otros usuarios para poder utilizar el simulador de manera conjunta, desde un dispositivo móvil.

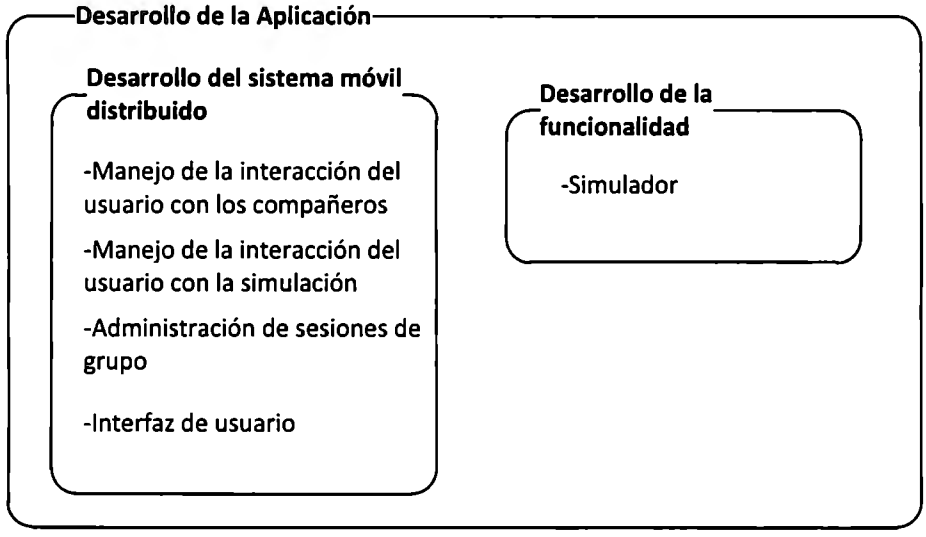


Figura 3. 1: Principales bloques del desarrollo de la aplicación

3.1. Descripción del Sistema a Modelar

Aquí Se describe a detalle la celda de ensamble de la cual se desarrolla el simulador de este proyecto. Se describen los módulos que la componen y el trabajo que realizan.

El sistema a simular es una celda de manufactura automatizada que pertenece al laboratorio de Ingeniería Industrial y de Sistemas del Instituto Tecnológico y de Estudios Superiores Monterrey, campus Ciudad de México. El cual es usado para fines didácticos y se muestra en la Figura 3. 2.

Figura 3. 2: Celda de ensamble para uso didáctico del laboratorio de IIS del ITESM-CCM

El sistema a modelar es, más precisamente, una celda de manufactura de ensamble sencilla, en la que se introducen unas piezas al sistema, a las cuales se les introducen un par de pernos, para posteriormente ser agrupadas en *pallets* de piezas de un mismo color (pueden ser amarillas o azules) y colocarlas en un almacén. Figura 3. 3 muestra los y los componentes mencionados.

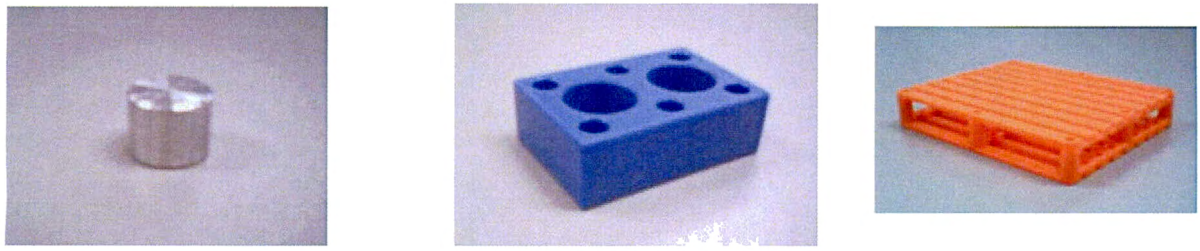


Figura 3. 3: Ejemplos de perno, pieza y pallet (de izquierda a derecha)

La celda de manufactura se compone de 5 módulos descritos a continuación, cuyo funcionamiento, al ser el código de dicha celda considerablemente extenso y no autodocumentado, tuvo que ser analizado por medio de observación directa:

Módulo 1: Este módulo es donde se reciben las pizas que llegan a la celda de manufactura y donde se les colocan los pernos. Para la modelación se consideró que, en este módulo, la pieza se encuentra en proceso desde que se le coloca el primero de dos pernos hasta que se le coloca el segundo y se coloca en la banda. Ver Figura 3. 4.

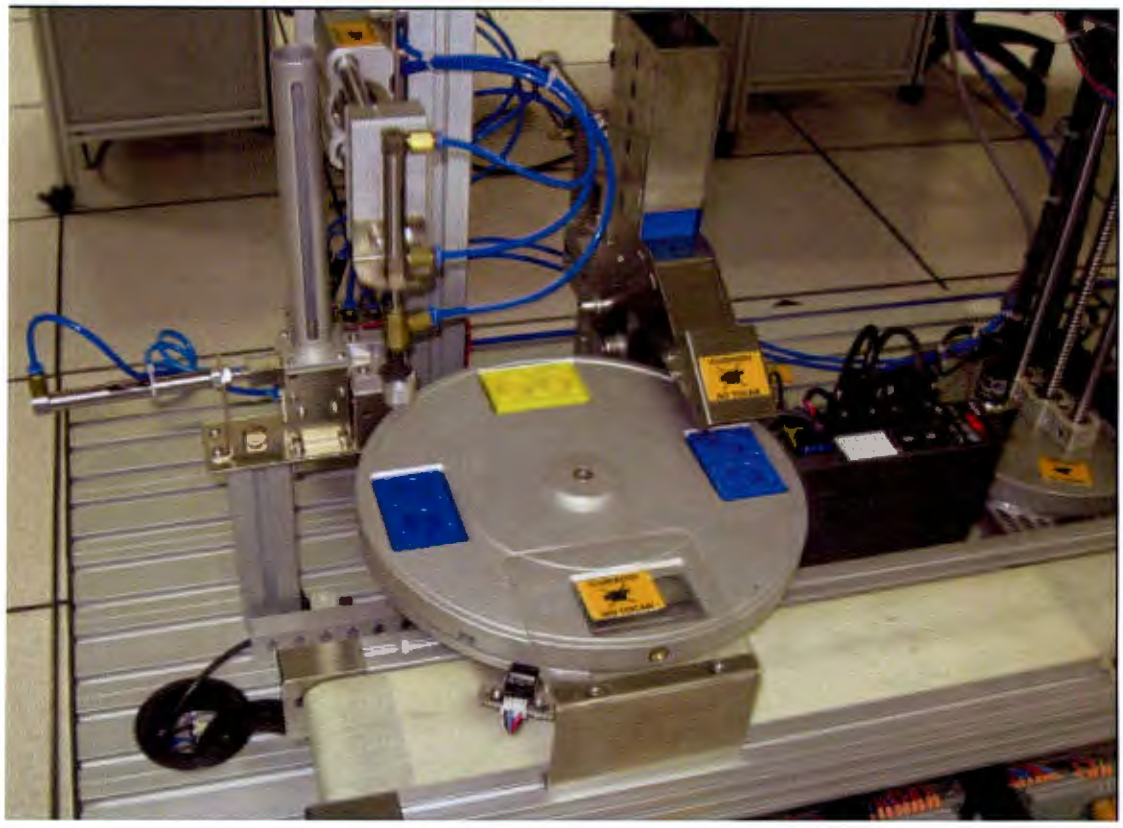


Figura 3. 4: Módulo 1 de la línea de ensamble

Módulo 2: Este módulo consiste en un brazo mecánico que toma las piezas, de una en una, para llevarlas de la banda de transporte donde las colocó el módulo 1 a otra banda distinta, la cual está conectada con el siguiente módulo. El tiempo que este módulo tarde en realizar su tarea dependerá del siguiente módulo, ya que sí alguna pieza tiene que esperar cierto tiempo para ingresar a este último, tendrá que esperar la misma cantidad de tiempo antes de cambiar a la siguiente pieza de banda (solo manda piezas al siguiente módulo cuando aquel está desocupado). La Figura 3. 5 muestra el módulo 2.

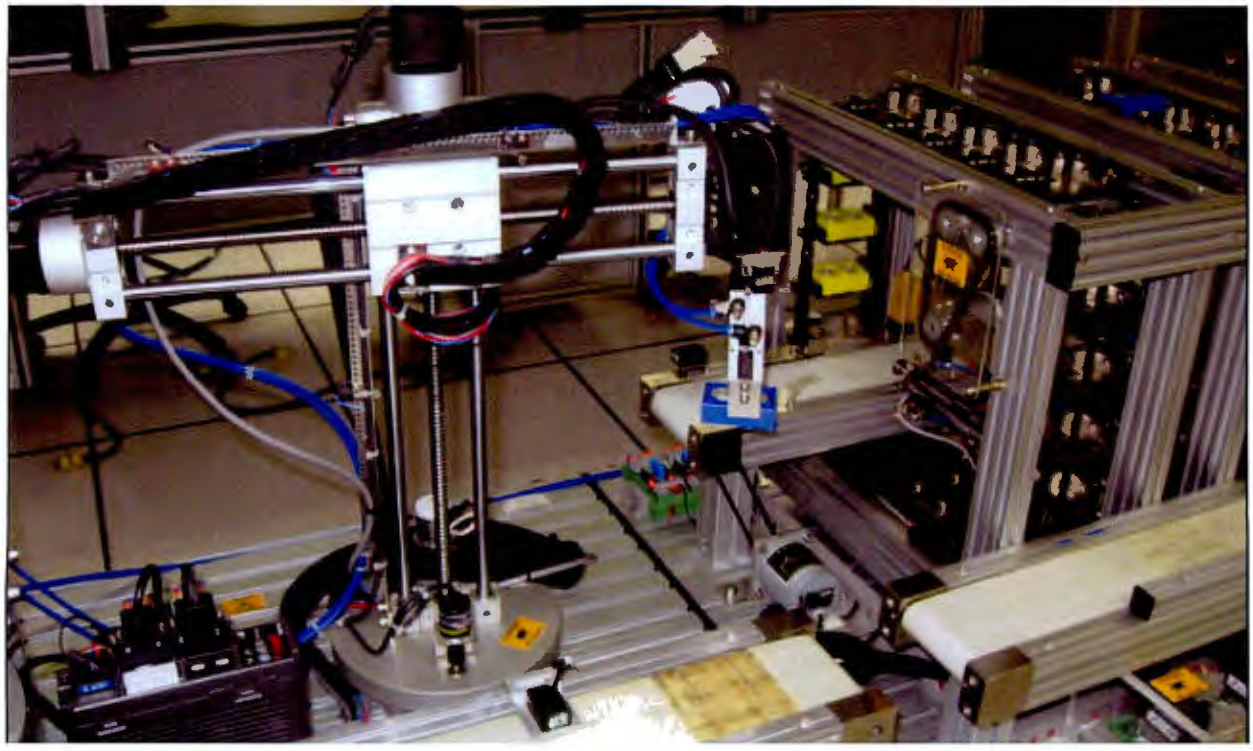


Figura 3. 5: Módulo 2 de la celda de ensamble

Módulo 3: En este módulo las piezas son colocadas en almacenes temporales, en forma de carrusel, asignados cada uno a un color determinado de piezas (el tercero de los tres es utilizado cuando cualquiera de los almacenes anteriores ha sido llenado y puede almacenar piezas de cualquier color). Cada almacén, al tener 10 piezas almacenadas, comenzará a sacarlas de una en una para que sean llevadas al siguiente módulo, excepto cuando alguno de los otros almacenes se encuentre sacando piezas o el siguiente módulo este agrupando un *pallet*, teniendo que esperar a que se terminen dichas actividades para proceder. En el caso en el que el almacén que se encuentra sacando piezas vuelva a tener diez piezas una vez que se haya terminado de formar el *pallet* con las piezas que esté sacando, si alguno de los otros almacenes tiene 10 piezas también, le sedera el turno de sacar piezas a este último. (Ver figura 3.6)

Figura 3. 6: Módulo 3 de la celda de ensamble

Los almacenes en forma de carrusel mencionados consisten en un conjunto de compartimientos individuales de piezas dispuestos en forma circular alrededor de un eje que girará cuando le sea necesario, ya sea para colocar un compartimiento ocupado en la única posición de salida (para sacar una pieza) o, para colocar un compartimiento vacío en la única posición de entrada (para introducir una pieza).

La velocidad con la que cada pieza sea colocada en su compartimiento de su carrusel correspondiente va a depender de en qué almacén sea ubicada (cada uno de ellos tiene una distancia diferente al módulo 2), y de la ocupación de dicho almacén (y a que la pieza tendrá que esperar a que el almacén gire hasta encontrar un lugar vacío en dicho almacén, este factor afectará también el tiempo de salida del almacén de las piezas, de modo similar).

En algunas circunstancias uno de los almacenes del módulo tres tendrá que recibir piezas en el momento en que se encuentra sacando un grupo de diez de estas, razón por la que se pueden dejar espacios vacíos en dicho almacén, la existencia de estos huecos puede afectar la velocidad con la que éste, al momento de sacar grupos de piezas, localice la siguiente pieza a sacar (el almacén tendrá que girar hasta encontrar un compartimiento ocupado para sacar la pieza que se ahí se encuentre). Por otro lado puede existir también la posibilidad de que, al llegar una pieza esta tenga que esperar a que su almacén gire hasta encontrar un hueco donde colocarla.

Módulo 4: Se encarga de crear *pallets* de dos pisos de 5 piezas cada uno y de colocar dichos *pallets* en el vehículo automatizado que los llevará al siguiente módulo. Este módulo tiene dos procesos: 1) En el que agrupa el primer piso del *pallet*. 2) En el que agrupa el segundo piso del *pallet* y se coloca el *pallet* completo en el vehículo que lo llevará al almacén final. La Figura 3. 7 muestra al módulo 4.

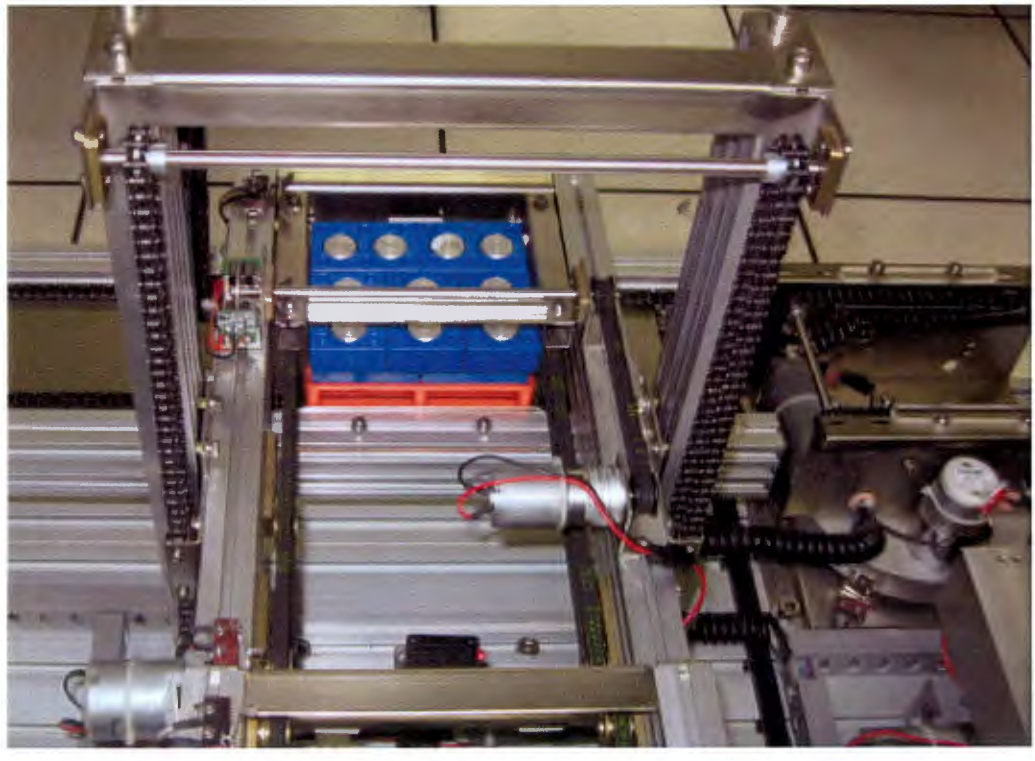


Figura 3. 7: Módulo 4 de la celda de ensamble

Módulo 5: Comprende un almacén final compuesto por 2 repisas y un vehículo automatizado que lleva cada pallet a un espacio dentro cualquiera de las dos repisas de acuerdo a un orden preestablecido. El tiempo que le toma a cada *pallet* finalizado llegar a su sitio en el almacén final dependerá de la ubicación que tendrá ese sitio y de las velocidades horizontal y vertical con las que funcione el vehículo automatizado. Ver Figura 3. 8.

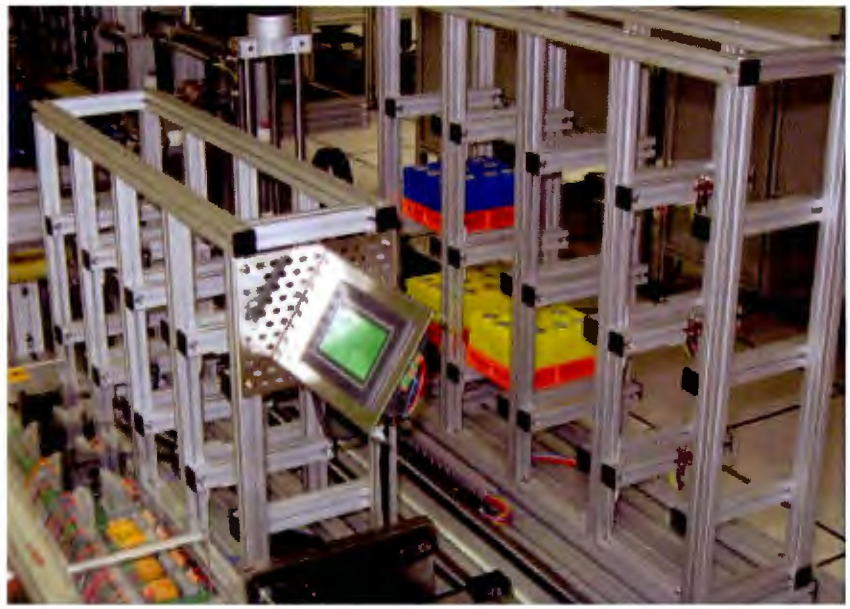


Figura 3. 8: Módulo 5 de la celda de ensamble

Secuencia de Operación de los Módulos: Las piezas recorren la celda de ensamble del módulo 1 al módulo 5 en orden numérico. La Figura 3. 9 muestra un plano de la celda de manufactura.

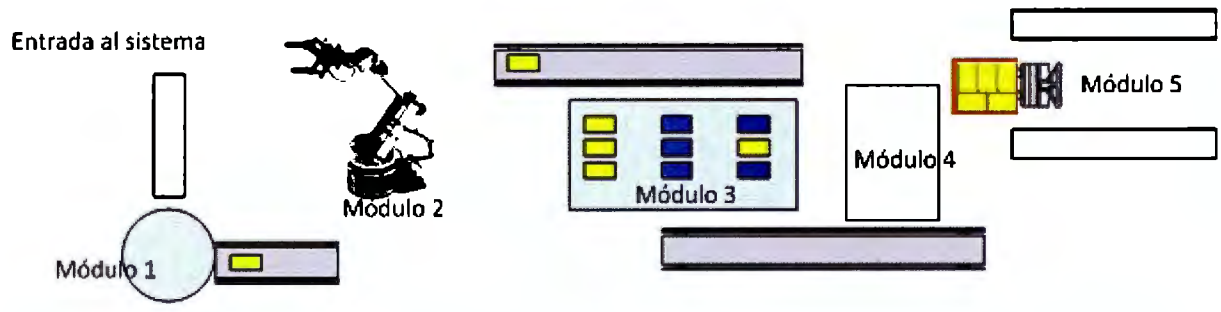


Figura 3. 9: Plano de la celda de ensamble

El diagrama mostrado en la Figura 3. 10 resume el trabajo que realizan en conjunto los diferentes módulos que componen la celda de ensamble.

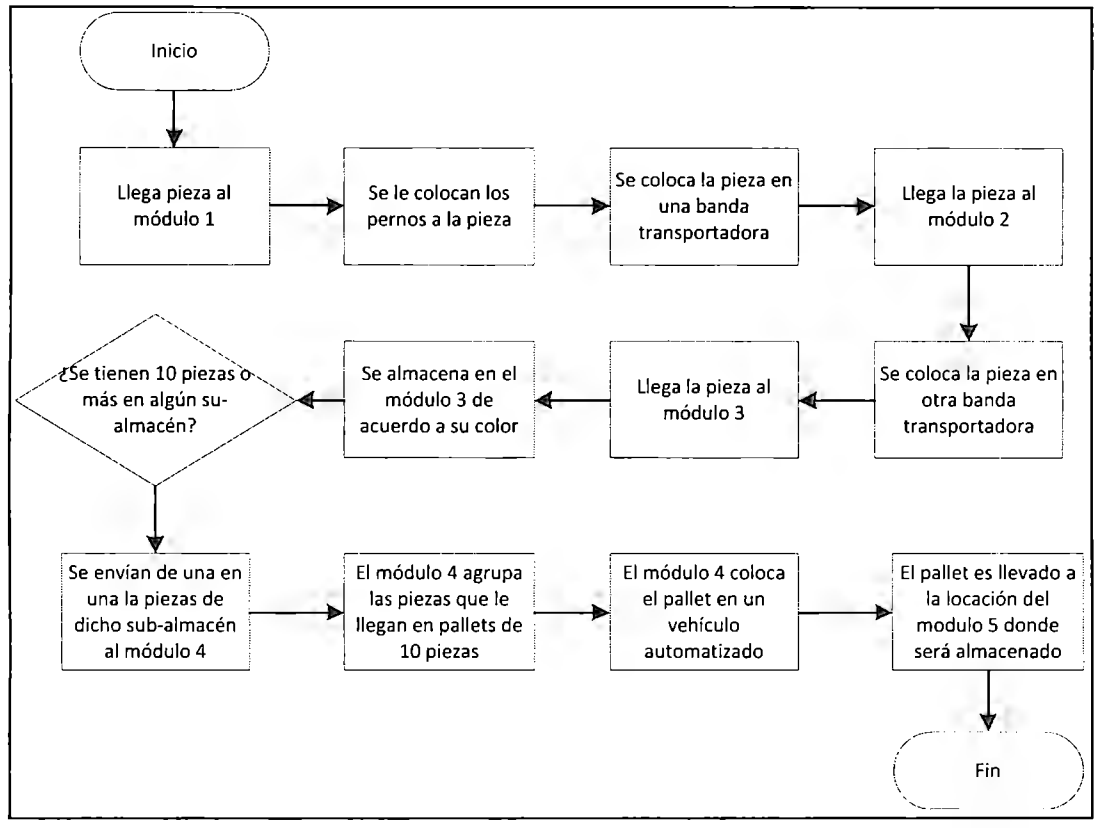


Figura 3. 10: Diagrama de flujo del funcionamiento de la celda de ensamble

3.1.1. Representación del Sistema como Modelo de Colas

A continuación se describe la representación del sistema como modelo de colas. A partir de este modelo se desarrolla un modelo más detallado de dicho sistema, a partir del cual se modelarán rasgos específicos del tipo de sistema al que pertenece.

Se definió a las piezas en las que se insertan los pernos y a los *pallets* que agrupan dichas piezas como los clientes que solicitan algún trabajo (e.g. recibir pernos, ser almacenados, ser cambiados de banda o ser agrupados en *pallets*). Los módulos mencionados en la sección anterior, que proporcionarían los trabajos anteriormente mencionados, serían entonces los servidores, cuyas colas representarían los artículos que estén en espera de recibir el trabajo que brinde el módulo

en cuestión. A Continuación se enlistan los servicios o trabajos que realizan cada uno de los servidores:

- Módulo 1: Colocar un par de pernos en cada pieza que reciban, las piezas que lleguen a este módulo serían colocadas en una cola con una política de tipo "la primera en llegar la primera en salir" (FIFO por sus siglas en inglés).
- Módulo 2: Cambiar las piezas de la banda a la que salen del módulo1 a otra. Su cola tiene una política de tipo FIFO.
- Módulo 3: A pesar de ser un almacén temporal, se consideró al servicio que brinda dicho módulo como la agrupación de las piezas para la formación de un *pallet*, de manera que, los diferentes sub-almacenes que lo componen se modelaron como colas con una disciplina especial que refleje la forma en la que las piezas van abandonando dicho módulo.
- Módulo 4: colocar piezas ya trabajadas en un *pallet.* En este módulo se considera que las piezas van llegando de una en una, y una vez realizado el trabajo, es decir, una vez que las diez piezas que componen el pallet se encuentran en éste, abandonan el módulo al mismo tiempo, en el mismo pallet, sin embargo, para este momento ya no se consideran piezas individuales sino componentes del *pallet que integran.*
- Módulo 5: Recibir los pallets ya agrupados que lleguen al almacén final. En este caso no se considera la existencia de ninguna cola ya que la línea de ensamble opera de tal manera que nunca se forma ninguna cola de espera en este módulo.

La población de clientes potenciales se considera infinita, ya que estos podrán llegar durante prácticamente cualquier intervalo de tiempo determinado por el usuario en base a una distribución de probabilidad escogida por él mismo.

En cuanto a la capacidad del sistema, existe un límite de capacidad en el módulo 3, una vez que se encuentren llenos los tres sub-almacenes que lo componen, entonces éste no pude recibir ninguna piza adicional, de recibir una pieza el tercer sub-almacén (el de las piezas mixtas) se mantendría girando buscando donde colocar la pieza, al no encontrar donde colocarla se

mantendría buscando de manera indefinida y quedaría detenida la línea de ensamble completa. Sin embargo con la configuración que tiene actualmente la celda es muy difícil que llegue si quiera a usarse el tercer sub-almacén, que es el de repuesto.

3.2. Requerimientos del sistema

Esta sección da una descripción de los usos que se le darán a la aplicación con motivo de determinar las características que ésta deberá tener. Después de realizar un análisis del sistema se determinó que, para complementar adecuadamente el uso de la línea de ensamble, y para alcanzar los objetivos previamente planteados para el proyecto, sería adecuado implementar el simulador en una aplicación que tenga las siguientes características principales:

- Funcionamiento de forma distribuida, de manera que los usuarios pudieran utilizar la misma instancia del simulador en modo multiusuario para poder configurarlo y ejecutarlo en equipos de trabajo. De este modo se podrían crear las prácticas colaborativas mencionadas en los objetivos del proyecto.
- Accesibilidad a través de dispositivos móviles.
- Ejecución de simulaciones de la línea de ensamble en tiempo real, bajo distintos tipos de escenarios.

Ya que los usuarios accederán de manera remota al simulador y lo utilizará en modo multiusuario, se determinó que los usuarios se debieran registrar, de manera remota, en un grupo multiusuario de trabajo y, junto con sus compañeros de grupo, pudieran determinar el valor de parámetros específicos de los diferentes módulos del simulador, para después ejecutar simulaciones de la línea de ensamble. Después de analizar el sistema se determinó que los usos que se le deben permitir al usuario serían los enlistados a continuación (el diagrama de casos de usos se muestra en la Figura 3. 11):

1. Los usuarios podrán trabajar en equipos.
2. Los alumnos deberán ser capaces de observar el funcionamiento de la línea de ensamble bajo diferentes configuraciones y tipos de demanda de trabajo (como pueden ser

diferentes intervalos de tiempo entre las llegadas de piezas a procesar, llegadas de dichas piezas en base a diferentes distribuciones estadísticas o diferentes mezclas de productos)

3. Los usuarios deberán ser capaces de intercambiar ideas con sus compañeros de grupo y manipular la configuración de la misma instancia del simulador, pudiendo ejecutar simulaciones en equipo y observar los mismos resultados.
4. Los usuarios deben poder observar el desarrollo de la simulación en tiempo real (o aproximadamente real).
5. Debe existir un registro de las simulaciones realizadas que pueda ser revisado por el profesor o inclusive por los mismos usuarios.
6. La aplicación deberá generar información que permita generar diferentes análisis estadísticos e índices de desempeño de la línea de ensamble en la simulación.

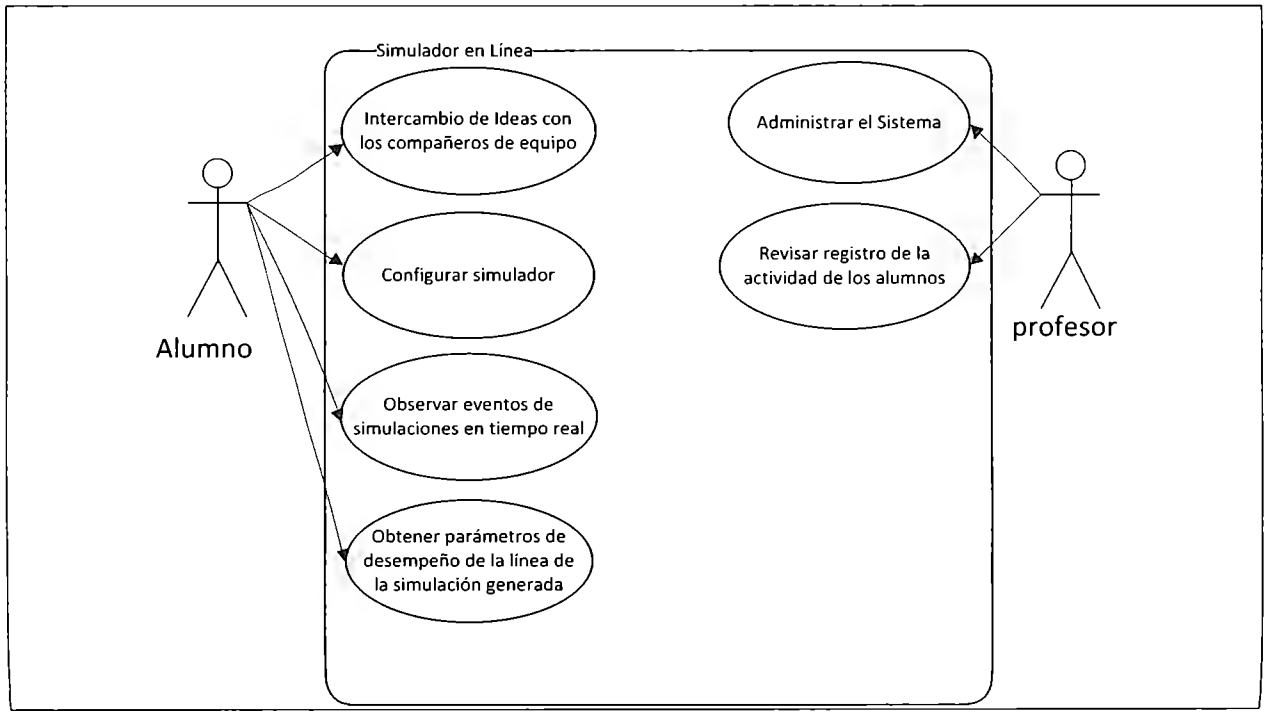


Figura 3. 11: Diagrama de casos de uso

3.3. Descripción del Funcionamiento de la Aplicación

En esta sección se detalla cómo es que el usuario va a acceder al simulador, y como va a interactuar con sus compañeros para configurar y ejecutar el simulador, así como ver una representación gráfica de dicha simulación.

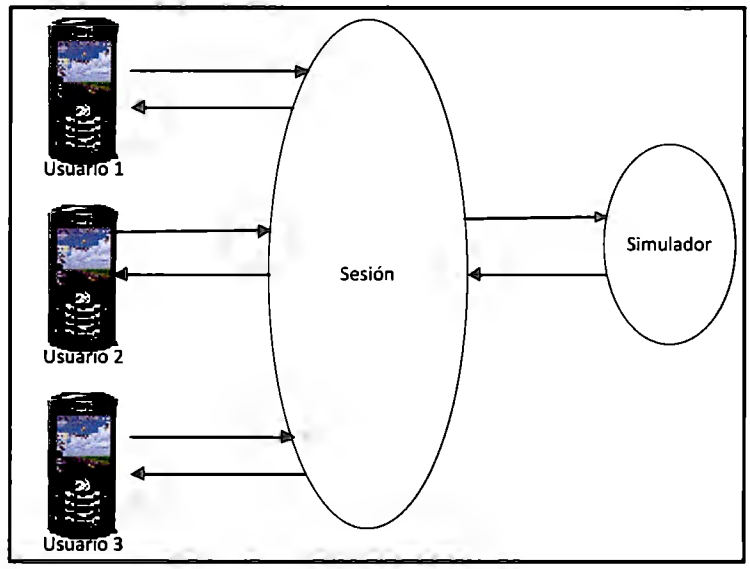


Figura 3. 12: Interacción entre el usuario y el simulador

Ya que se especificó que la aplicación deberá proveer la opción de ser usada en modo multiusuario, se determinó que el usuario ingresara a una sesión de su grupo de trabajo, y que en dicha sesión configurara y ejecutará junto con sus compañeros de equipo una misma instancia del simulador de la línea de ensamble, es decir, por cada sesión de equipo se crea una instancia del simulador (ver Figura 3. 12). El usuario tendrá la opción de ingresar a una sesión previamente creada o crear una sesión nueva (ver Figura 3. 13).

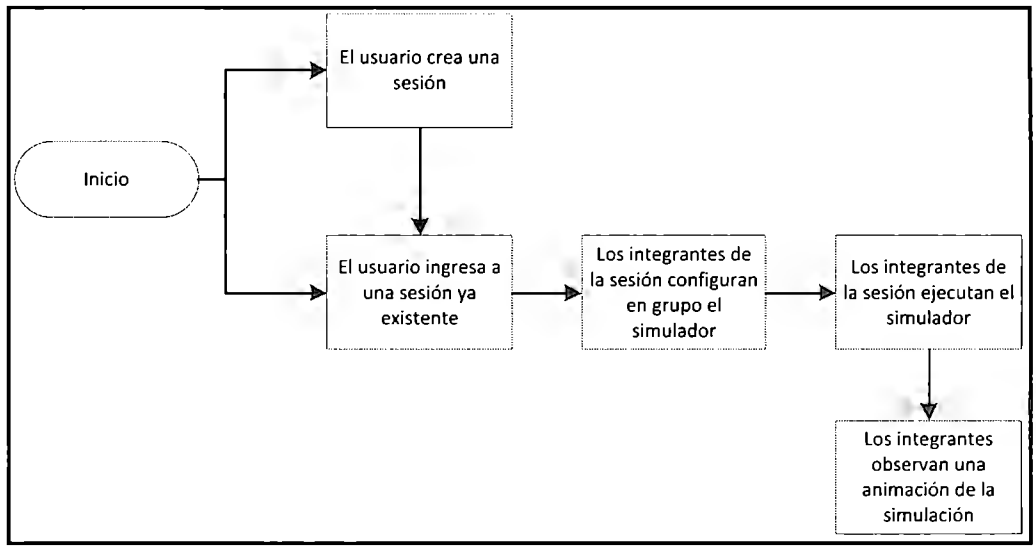


Figura 3. 13: Diagrama del uso del simulador en sesiones

Para configurar el simulador, el usuario podrá seleccionar un módulo, una vez seleccionado, el usuario podrá cambiar los parámetros que de dicho módulo sean modificables, como puede ser la velocidad en la que se realiza un proceso.

En el caso del primer módulo, lo que se va a modificar va ser el tiempo promedio entre llegadas de piezas a este, que van a ser los tiempos promedio entre llegadas al sistema, así como la distribución de probabilidad que tengan las mismas llegadas. En caso de que el usuario escoja un tiempo de llegadas constante, el tiempo entre llegadas constante será el valor que se le dé al tiempo entre llegadas promedio. En el caso del módulo 4, las variables que se podrán modificar serán: el tiempo de su proceso 1 (la velocidad con la que se forma el primer piso del *pallet*) y el tiempo de su proceso 2 (la velocidad con que se forma el segundo piso del *pallet*), por otro lado, las variable modificables del módulo 5 son las velocidades tanto vertical como horizontal con que se desplaza el dispositivo que coloca los pallets en dicho módulo, que es el almacén final.

Para coordinar la configuración del simulador entre los diferentes usuarios, cada módulo va a poder ser seleccionado solamente por un usuario a la vez, al estar seleccionado un módulo, éste

dejará de estar disponible para cualquier otro usuario hasta que el usuario que lo tenga seleccionado lo cambie por otro (ver Figura 3. 14).

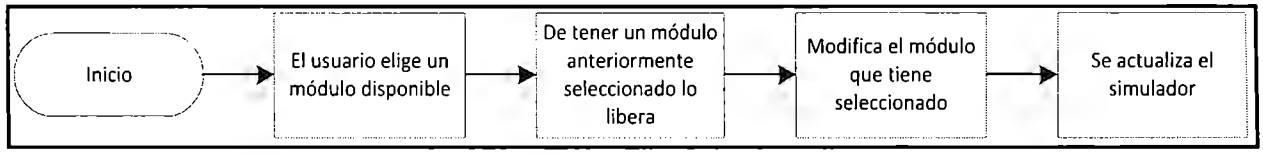


Figura 3. 14: Diagrama de flujo de la configuración del simulador en sesiones de varios usuarios

Para coordinar la ejecución del simulador, los usuarios podrán modificar su estatus de listo a no listo o viceversa, el estatus también indicará qué módulo tiene actualmente seleccionado el usuario. Una vez que todos los usuarios estén listos en su estatus, cualquier usuario podrá seleccionar la opción de "iniciar simulación" la cual hará funcionar el simulador para posteriormente hacer visible la representación gráfica de los eventos generados en la simulación a cada usuario. El simulador no se ejecutará si alguno de los usuarios no se encuentra listo en su estatus (ver Figura 3. 15).

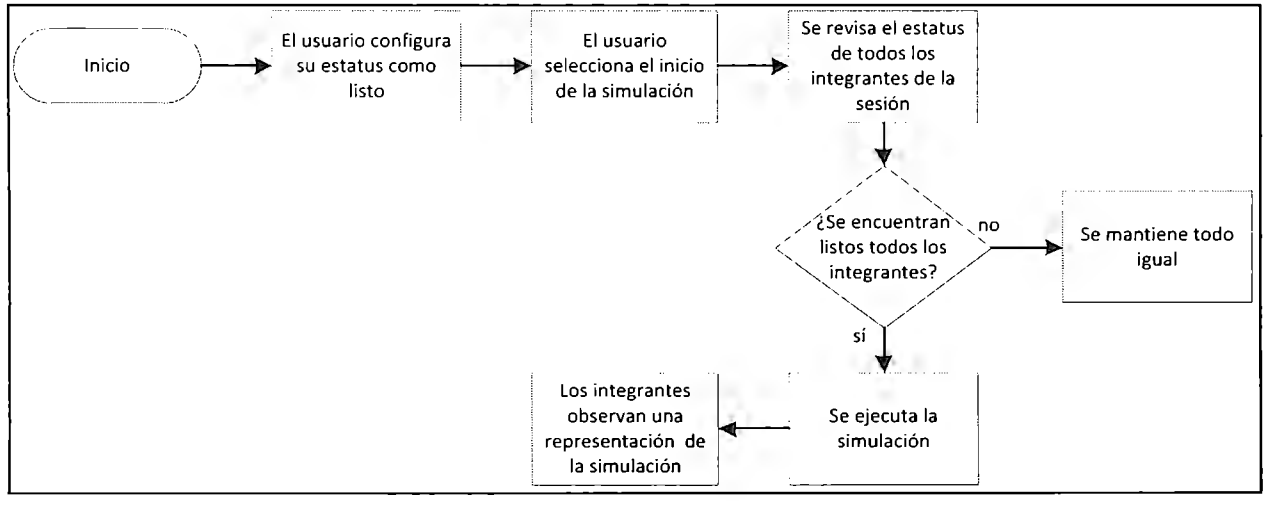


Figura 3. 15: Diagrama de flujo de la ejecución de simulaciones en sesiones de varios usuarios

3.4. Características del Simulador

En esta parte se describen las características especiales que ofrecerá el simulador al usuario. La Figura 3. 16 muestra las interacciones que el usuario podrá tener con el simulador.

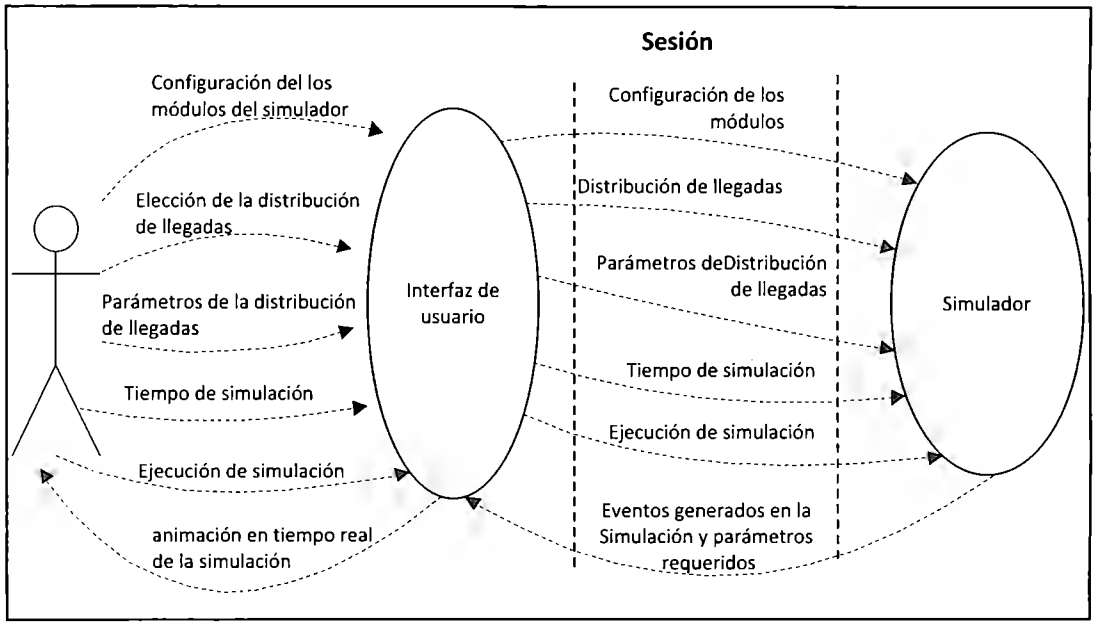


Figura 3. 16: Interacción entre el usuario y el simulador

El diseño de la aplicación se realizó de tal forma que el usuario pueda seleccionar una distribución estadística para el tiempo entre llegadas al sistema, así como determinar los parámetros de dicha distribución estadística, como es el tiempo promedio entre llegadas, además de las modificaciones que podrá realizar a algunos módulos de la línea de ensamble, dicha configuración la determinará tanto para piezas de color azul como para piezas color amarillo. El usuario también podrá determinar el tiempo de operación que será simulado. La simulación generará como salida una lista de eventos, dichos eventos servirán para que en otra parte de la aplicación se generen animaciones de las simulaciones y contendrán la siguiente información:

- El tiempo en que se dio el evento.
- El módulo en que se dio el evento.
- La pieza o el *pallet* involucrado en el evento.
- El tipo de evento (llegada o salida).

Como resultado de la simulación, el usuario podrá observar la animación de la simulación generada, la cual será en tiempo real, aunque podrá acelerar la velocidad a la que ésta se desarrolle, de manera que pueda observar el comportamiento del sistema de una manera más

práctica y simular cantidades de tiempo de operación relativamente largos, hasta que se llené el módulo 5, que es el almacén final de *pallets*. Adicionalmente se generará un registro estático en archivo de texto de los diferentes eventos que se generaron en la simulación, en el servidor de manera que pueda ser observado tanto por el mismo usuario como por el profesor.

Los eventos y los diferentes parámetros relevantes que son directamente observables de la línea de ensamble quedarán almacenados temporalmente en la memoria del servidor, como son la cantidad de artículos en cada módulo, la posición de cada *pallet* en el almacén final, así como la posición que ocupa cada pieza en el módulo 3, compuesto por tres sub-almacenes giratorios, con dicha información y con la lista de eventos generada es relativamente fácil crear animaciones detalladas de la simulación, así como generar diferentes estadísticas detalladas del desempeño de la línea de ensamble en la simulación de ser necesarias a futuro.

3.5. Diseño de la Funcional del Simulador Móvil Distribuido

En esta sección se habla de la funcionalidad central de la aplicación, que en este caso es el simulador como tal. En esta sección se describe cómo se modeló la línea de producción a simular, así como el diseño de esta parte de la aplicación.

Se describen el patrón y la estructura que se seguirán para el desarrollo del simulador, así como las herramientas a utilizar para generar su código. Posteriormente se describe la estructura que se le dio a la lógica del simulador. El modelo desarrollado en esta sección es un modelo más específico usado para representar características específicas de las líneas de producción, similares a la celda de ensamble del Tecnológico de Monterrey, de manera abstracta, a partir de este modelo y del modelo de colas de la sección 3.2.1, se generará un modelo a detalle de la celda que se va a simular en la sección 4.

En la siguiente sección se describirá el paradigma que se siguió para generar un modelo de simulación del sistema a simular, esto en base al modelo de colas de la sección anterior, el

modelo de simulación servirá para representar el sistema con motivo de realizar simulaciones dinámicas discretas.

3.5.1. Selección del paradigma de Modelación de Sistemas Dinámicos Discretos

En esta sección se describe la elección del paradigma de modelación de sistemas dinámicos discretos a seguir para la modelación a detalle del sistema.

Se eligió desarrollar la aplicación en base al paradigma basado en eventos ya que, al usar menor cantidad de recursos de memoria y procesamiento en su implementación en software, en comparación con el paradigma orientado a procesos, se obtiene una mayor flexibilidad para la implementación de la lógica funcional en dispositivos móviles (con menores capacidades de memoria y procesamiento), y de forma distribuida (que implicará el manejo de un mayor número de hilos de ejecución).

Al implementar un simulador modelado en base al paradigma orientado a procesos se necesitaría hacer uso de múltiples hilos de ejecución, lo cual, como se ha mencionado en la sección 2 de esta tesis, repercute en una ejecución más lenta, en comparación con los simuladores basados en el paradigma orientado a eventos, lo cual no resulta deseable, especialmente para simuladores en línea o distribuidos. El mismo hecho de implementar el simulador en línea (en modo multiusiario) o de manera distribuida implica el uso de hilos de ejecución adicionales.

El uso de múltiples hilos de ejecución puede tener resultados variados en diferentes plataformas (Java por lo general utiliza hilos de ejecución reales del sistema) por lo que el desarrollo mediante el paradigma orientado a procesos puede resultar complicado, además de ser propenso a generar puntos muertos (*dead locks*). En el caso de dispositivos con capacidades de memoria limitadas, el uso de multiples hilos de ejecución resulta ser una opción poco atractiva debido al extenso uso de recursos que se utilizarían. Aún al implementar la lógica funcional en una arquitectura cliente-servidor, al contemplar el uso de la aplicación por parte de varios grupos de

usuarios (exigiendo una cantidad probablemente extensa de procesamiento), el que el paradigma orientado a eventos sea más rápido y eficiente puede ser tomado como un factor importante a considerar.

Sin embargo, se terminó utilizando más bien una versión modificada del paradigma orientado a eventos, ya que, para eventualmente implementar la lógica funcional de manera distribuida entre varios dispositivos móviles de ser necesario, se trató de desacoplar las clases que componen el simulador, razón por la que, a pesar de que se utilizan eventos, se evitó crear una clase que controlara todas las acciones que se daban en el simulador, y se desarrollaron las diferentes clases que representan los módulos del simulador de tal forma que se coordinaran solas, por medio de sus propios métodos.

3.5.2. Selección de Herramientas para la Implementación Funcional del Simulador

Habiendo descartado previamente el hacer uso de lenguajes de simulación debido a que no brindan facilidades para crear simuladores en línea de tipo multiusuario, y a que se requeriría destinar tiempo para aprenderlos, se decidió únicamente utilizar las APIs disponibles en el kit de desarrollo de Java (JDK). Sin utilizar además ningún marco de trabajo especial para el desarrollo de simuladores, debido a que, para implementar aplicaciones hechas en Java en dispositivos móviles es comúnmente necesario hacer modificaciones al código para que se acople a la edición micro de Java que dichos dispositivos utilizan, ya que esta utiliza una versión reducida de las APIs disponibles para la versión estándar de Java. De modo que sería necesario conocer a fondo el código generado por el marco de trabajo en cuestión para poder ajustarlo a Java Microedition, de manera que esta opción ya no resultaría tan práctica.

Por otro lado se estimó que la implementación sin el uso de ningún marco de trabajo podría requerir una cantidad de tiempo similar a la que se requeriría para aprender a utilizar adecuadamente algún marco de trabajo para el desarrollo de simuladores, siendo que la mayor parte del trabajo de desarrollo está relacionado con las particularidades del sistema a modelar.

Además, el conocer a detalle el código a utilizar y poder modificarlo permitiría mayor flexibilidad de ser necesaria.

3.5.3. Modelación del Sistema de Producción

A continuación se describe como se modelaron las líneas de producción en general. Como se mencionó en la sección 3.5.1, para eventualmente implementar la lógica funcional de forma distribuida, en caso de que se tomara esa decisión, se decidió desarrollar el simulador en base a eventos, pero sin la utilización de una clase que controlara todas las acciones que se dieran en el simulador, desarrollando las clases que representaban los módulos del simulador de tal forma que se pudieran coordinar entre ellas, por medio de sus propios métodos. De modo que la coordinación entre los diferentes módulos fue un factor importante a considerar en el desarrollo de dichas clases, lo cual aumentó su complejidad.

En general, el simulador se basa en objetos de tipo módulo (**Module**), los cuales van a representar los diferentes módulos del simulador, estos van a recibir y enviar objetos de tipo **DynamicEntity**, los cuales, por su parte, van a representar "entidades desplazables o dinámicas" del sistema (como pueden ser piezas individuales o *pallets* que agrupen varias piezas). Los módulos van a crear objetos de tipo evento (**Event**) cada vez que ingrese o salga de ellos una pieza, estos objetos estarán compuestos por un objeto **DynamicEntity** (la pieza o pallet involucrada en el evento), el nombre del módulo donde se dio el evento y el tiempo en que ocurrió.

Se modelaron dos tipos de eventos, eventos de tipo **ArrivalEvent** (los cuales van a ser instanciados por los módulos cada vez que estos reciban una entidad dinámica) y eventos de tipo **DepartureEvent** (los cuales van a ser instanciados por los módulos cada vez que estos saquen una entidad dinámica).

Los módulos registrarán todos los eventos que instancien en un registro global perteneciente a un objeto llamado **schedule**, que se encargará del registro de eventos a lo todo lo largo de la simulación y será accesible a todos los módulos del simulador. De este modo se pueden ordenar los eventos generados en los diferentes módulos (de forma independiente) para simular un trabajo simultaneo de dichos módulos (ver Figura 3. 17). Este enfoque puede resultar benéfico en el caso de simulaciones distribuidas al evitar la necesidad de que componentes distintos situados en equipos distintos se coordinen para su ejecución de forma remota.

Además, el generar una lista de eventos como resultado de la simulación permite separar la generación de la lógica funcional (de la cual se haría cargo el simulador) de la generación de la presentación que únicamente tendría que generar la animación a partir de la lista de eventos sin tener su desarrollador que involucrarse con el código del simulador. En el caso de los almacenes pudiera ser necesario el uso de información adicional, sin embargo esta pudiera ser dispuesta igualmente a través de listas.

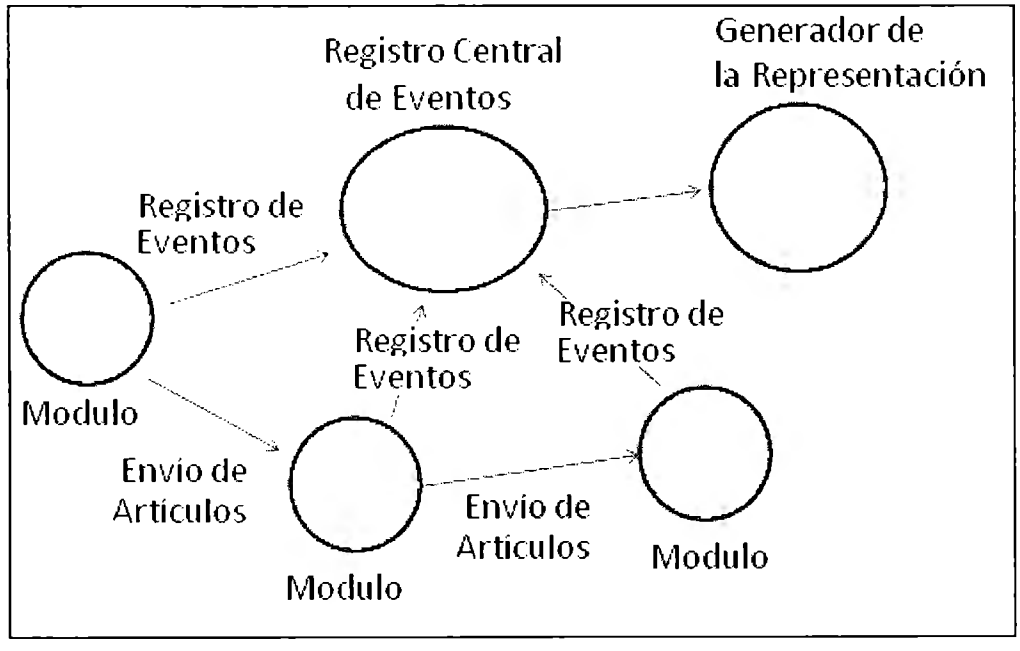


Figura 3. 17: Diagrama general del funcionamiento del simulador

La principal ventaja que presenta esta modificación al método orientado a eventos es que permite, en el caso de sistemas con una estructura "lineal", en el que no se considere el regreso

de piezas a módulos anteriores, una visualización más sencilla del código, de manera que resulta más fácil su desarrollo, su mantenimiento y la realización de pruebas que se realicen del simulador.

Lo anterior debido a que se reducen las circunstancias en las que se intercalan métodos de diferentes módulos en el flujo de ejecución de la aplicación, es decir, se limita el número de diferentes módulos implicados en los métodos que se encuentran en el stock en todo momento. De este modo resulta más fácil el análisis de las posibles fallas generadas en las pruebas, y la programación.

Para llevar un control del tamaño de sus colas y de ese modo determinar el tiempo de salida de los artículos que reciben, los objetos tipo module contarán con elementos para el almacenamiento de entidades dinámicas (colas), mediante los cuales podrán determinar cuántas entidades eran atendidas en un módulo o se encontraban en la cola de éste, esperando a ser atendidas, al momento en que llegue a él cada una de las piezas que le sean enviadas, para de este modo calcular el tiempo en el que saldrán del módulo las piezas recién llegadas. Por ejemplo, si una pieza p llega en un tiempo t=60 a un módulo que tarda 20 unidades de tiempo en trabajar cada pieza, y en el que se encuentran dos piezas. La primera siendo atendida y la segunda esperando a ser atendida, con tiempos t de salida programados en t=65 y t=85 respectivamente, entonces se puede determinar el tiempo de salida de la pieza recién llegada (en este caso el tiempo resultante de salida es de 85+20=105).

Cada vez que llegue una pieza a un módulo se retiran las piezas que tienen un tiempo de salida menor al tiempo de llegada de dicha pieza del elemento donde se registran las piezas actualmente presentes en el módulo, para de este modo determinar cuál era el tamaño de la cola de espera a la que llega y, de ese modo, cuál es su tiempo de salida para crear el evento de salida correspondiente.

Cada que un módulo saque un artículo o *pallet* de sus elementos de almacenaje creará un objeto tipo **DepartureEvent**, el cual registrará en la lista global de eventos y a partir del cual generará una llegada en el siguiente módulo. Los módulos se encargan de generar las llegadas al siguiente módulo al llamar a su método **arrival**, el cual va a pedir como argumentos el elemento que llega y el tiempo en el que llega este, para a partir de estos crear y registrar objetos de tipo **ArrivalEvent** en la lista de eventos global y generar a su vez las llegadas del siguiente módulo.

La Figura 3. 18 muestra un diagrama de flujo del funcionamiento general de los objetos tipo Module, en el cual se puede observar cómo son retiradas las piezas cuyo tiempo de salida dT es menor al tiempo aT en que llega la pieza, de modo que se pueda determinar su tiempo de salida dT a su vez. También se puede observar cómo son creados eventos de entrada o de salida según piezas llegan al módulo o lo abandonan.

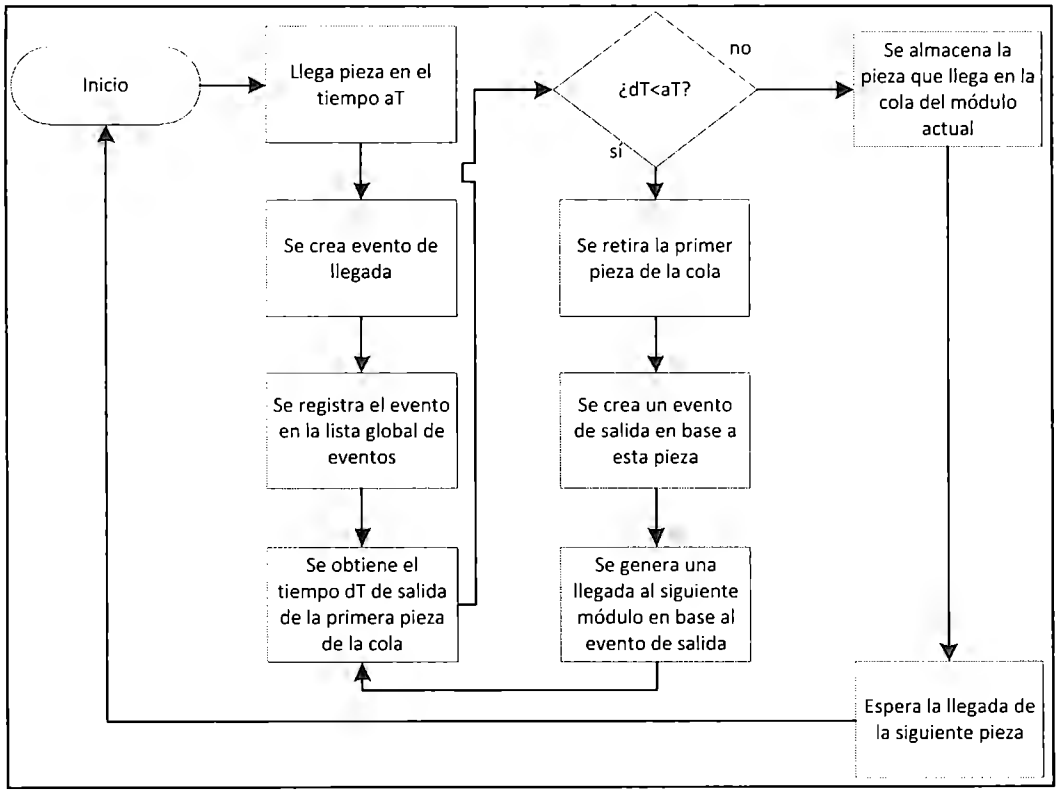


Figura 3. 18: Diagrama de flujo para los objetos tipo módulo

El usar objetos de tipo **DepartureEvent** para generar llegadas al siguiente módulo, en vez de invocar el método **arrival** de este último directamente permite crear una lista de eventos de salida para enviar su contenido al siguiente módulo hasta que el módulo actual haya terminado de recibir piezas o *pallets,* para de este modo ejecutar la lógica de cada módulo del simulador por separado.

Sin embargo no todos los módulos se pueden modelar de esta manera por lo que se consideraron 2 tipos de módulos, los módulos de tipo "push" y los de tipo "pull". Los módulos tipo "push" se definieron como aquellos que, una vez recibida y procesada una pieza, la envían al siguiente módulo, donde puede, dependiendo de la velocidad con la que se reciban y procesen las piezas, ser atendida directamente o almacenada temporalmente en una cola. Los de tipo "pull" se definieron como aquellos que mandan una pieza al siguiente módulo únicamente cuando éste se encuentra desocupado, es decir, este tipo de módulos no manda piezas a colas de espera, si el siguiente módulo se encuentra procesando una pieza, entonces se espera a que termine de procesarla para mandarle la siguiente. (Ver Figura 3. 19)

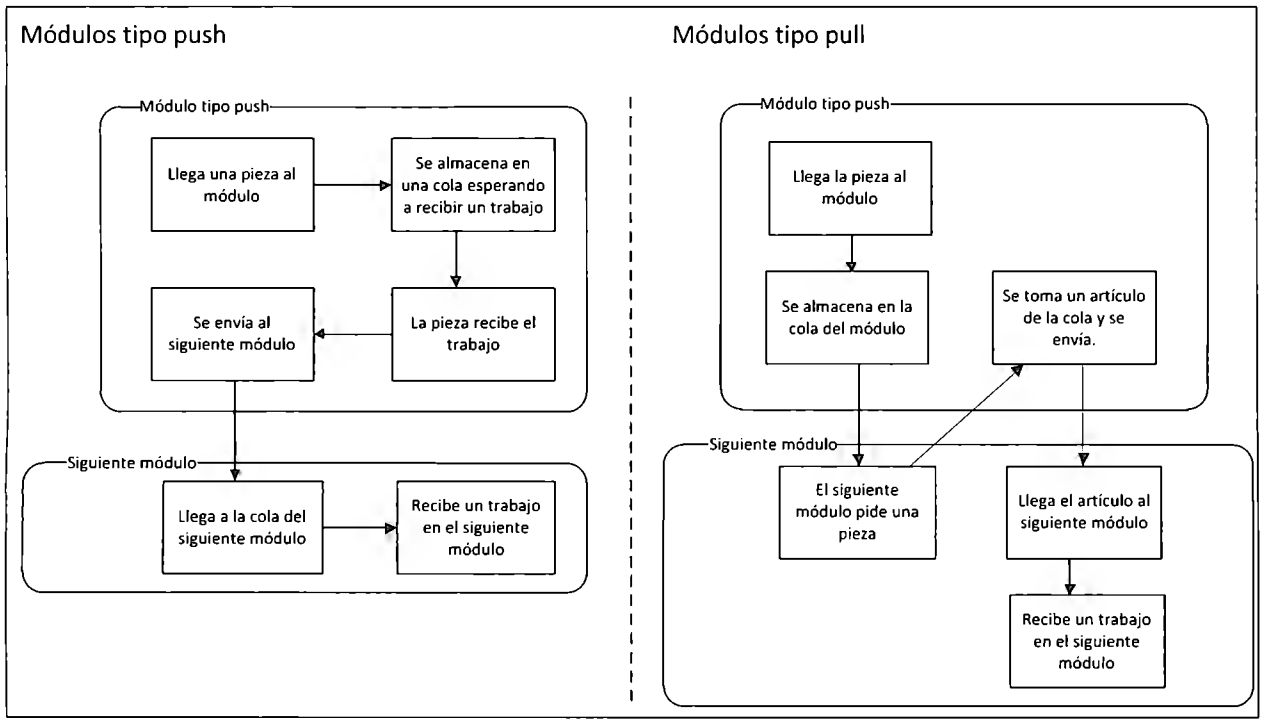


Figura 3. 19: Comparación entre los módulos tipo push y los de tipo pull

La diferencia entre los módulos de tipo "push" y los de tipo "pull" es relevante para determinar la secuencia en que se llevan a cabo las actividades dentro de cada módulo. En el caso de los módulos de tipo push, como es necesario que manden las piezas al siguiente módulo independientemente del tiempo que este último tarde en recibirlas y procesarlas, generan una lista de eventos de tipo **DepartureEvent** (que incluyen el objeto tipo pieza o pallet que sale y el tiempo en que abandona el módulo), de la cual envía su contenido al siguiente módulo una vez que haya terminado de procesar todas la piezas que le llegaron durante toda la simulación, esto permite lidiar más fácilmente con los tiempos de cada módulo y localizar cualquier problema en el código más rápido, al desarrollarse la simulación, en cierto grado, módulo por módulo, simplificando el desarrollo al evitar colocar métodos de diferentes módulos en el stack. La Figura 3. 20 ilustra el funcionamiento general de los módulos tipo "*push*" mediante un diagrama de flujo.

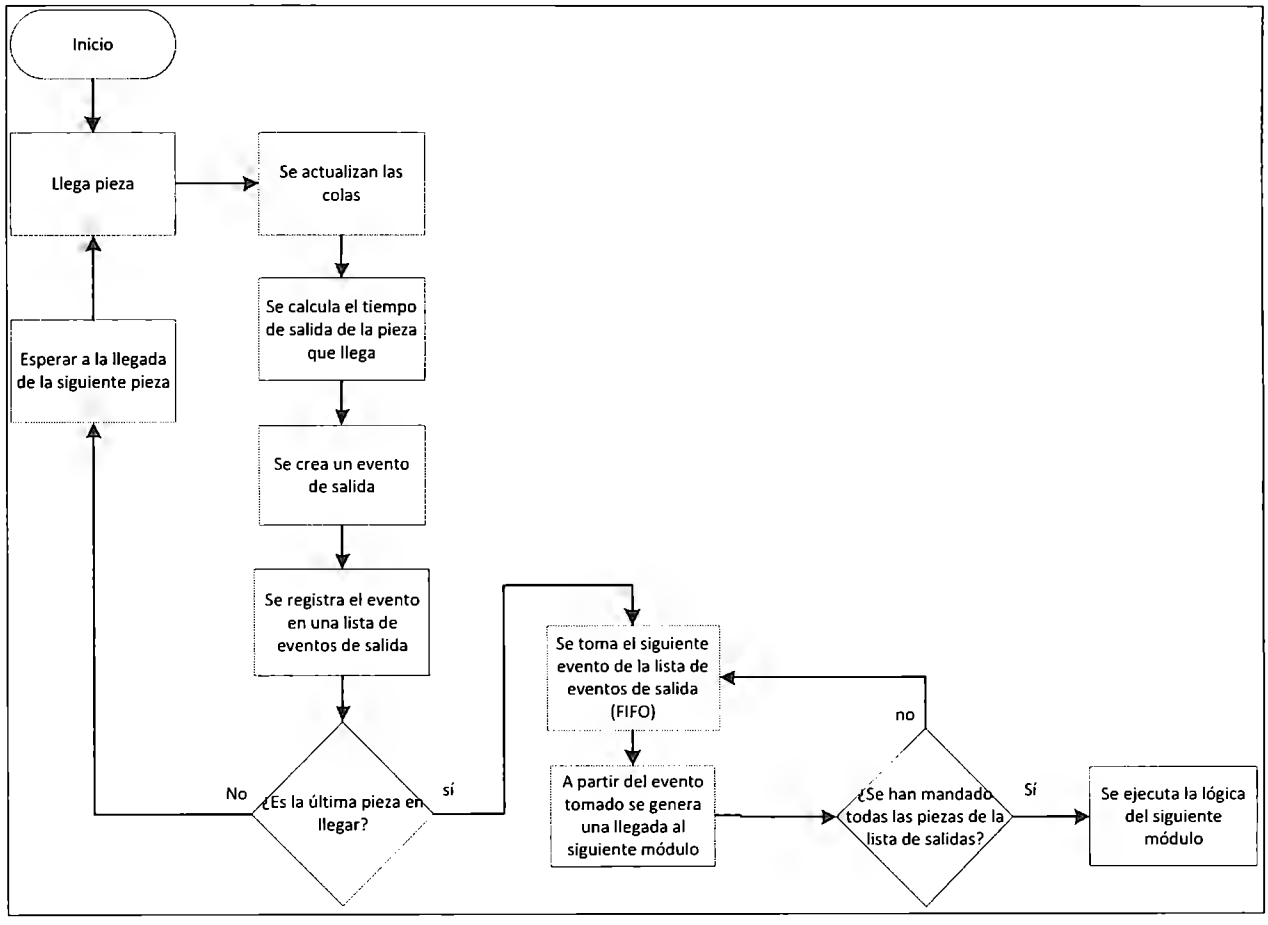


Figura 3. 20: Diagrama de flujo para los módulos tipo "push"

Sin embargo, en ocasiones es necesario invocar métodos de un módulo mientras se ejecuta la lógica del módulo anterior, como sucede en el caso de los procesos de tipo "pull", en los cuales es necesario llamar directamente al método de llegada del siguiente módulo y que este devuelva el tiempo en que dicho artículo fue procesado o almacenado (en el caso de los almacenes), para programar, o reprogramar, la salida del siguiente artículo. Los módulos tipo "pull", en lugar de almacenar los objetos de tipo **DepartureEvent** para enviarlos al siguiente módulo hasta que hayan terminado de recibir piezas, generarán directamente, a partir de dichos eventos de salida, llegadas al siguiente módulo, como se muestra en la Figura 3. 21.

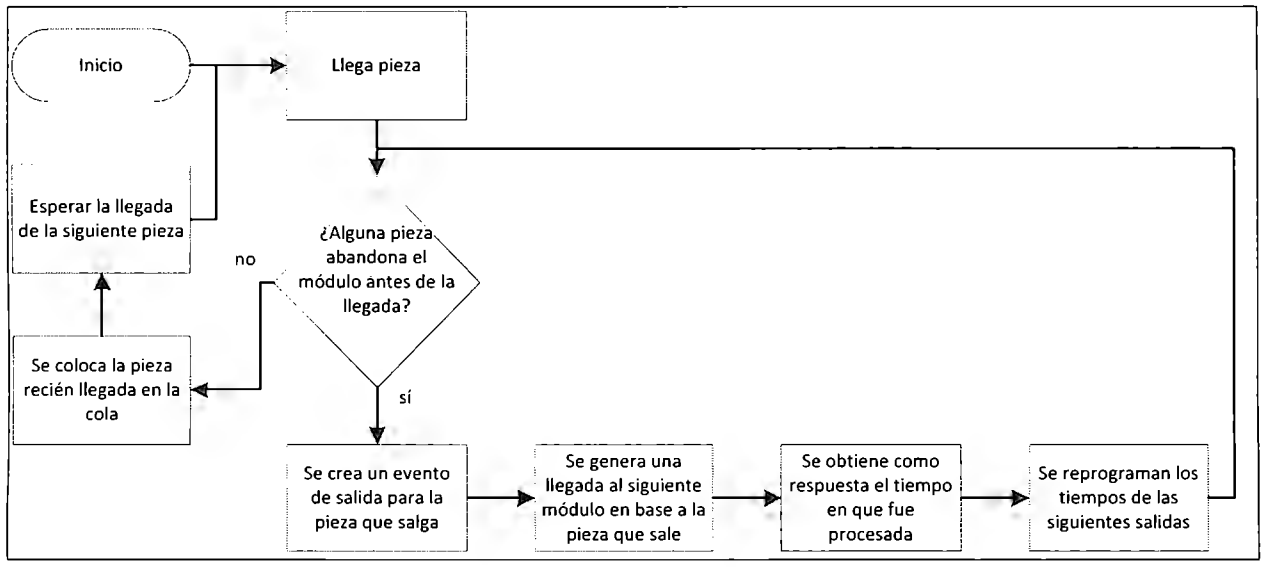


Figura 3. 21: Diagrama de flujo para módulos tipo "pull"

Los módulos de tipo almacén se modelaron como un caso especial de módulo de tipo "push", en el cual el tiempo de salida de los artículos estaría determinado por alguna condición específica, como pudiera ser la cantidad de artículos almacenados, y cuyos tiempos de llegada de piezas sería afectado por variables como la distancia al lugar de almacenamiento o la cantidad de entidades previamente almacenadas. La Figura 3. 22 ilustra el funcionamiento general de los módulos tipo almacén mediante un diagrama de flujo, en éste se puede observar que, al igual que en los módulos tipo "push", los eventos de tipo salida van a ser almacenados en una lista, a partir de la cual se generarán llegadas al siguiente módulo una vez que haya llegado la última pieza. Sin embargo, a diferencia de los módulos de tipo "push", en los que se retiraban las piezas con tiempo menor al tiempo de llegada de las piezas que llegan, los módulos tipo almacén, retirarán las piezas de acuerdo a alguna condición determinada, como puede ser la cantidad de piezas que tienen almacenadas.

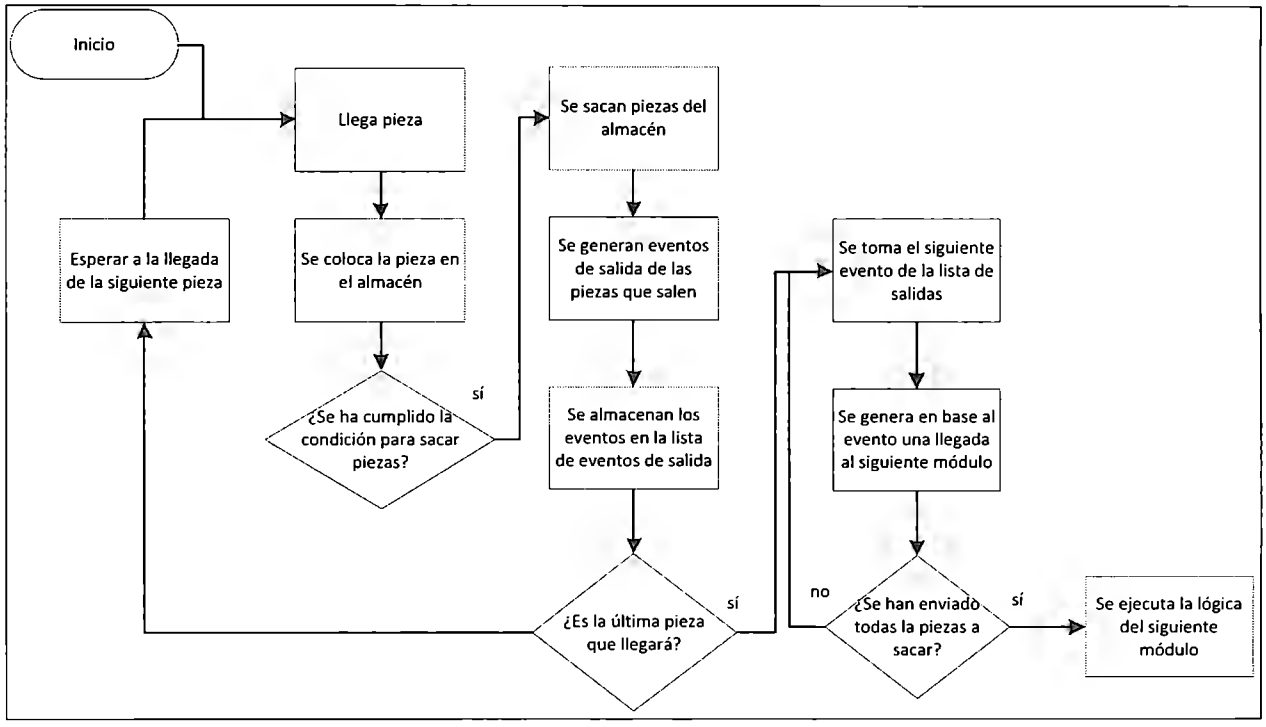


Figura 3. 22: Diagrama de flujo para los módulos tipo almacén

Otra tipo especial de módulo *Push* es el llamado *Palleticer*, este va a recibir piezas tipo artículo y va a sacar *pallets* (compuestos por varios artículos individuales).

3.5.4. Secuencia del Funcionamiento del Simulador

En esta sección se explica cómo es que las diferentes clases del simulador (e.g. las clases tipo Module) van a interactuar entre sí de manera que simulen el funcionamiento de una línea de producción automática, y cómo es que se activan sus respectivos procesos.

El simulador va a tener una clase principal llamada **Handler**, la cual va a generar las instancias de tipo **Module** utilizadas por el simulador, va a contener un conjunto de métodos de tipo **setter** para la configuración de estas, va a instanciar la clase tipo **Schedule**, donde se llevará el registro global de eventos del simulador, y va a tener un método para iniciar la simulación. De modo que

las demás partes de la aplicación, para interactuar con el simulador, utilizarán una instancia de esta última clase. (Ver Figura 3. 23)

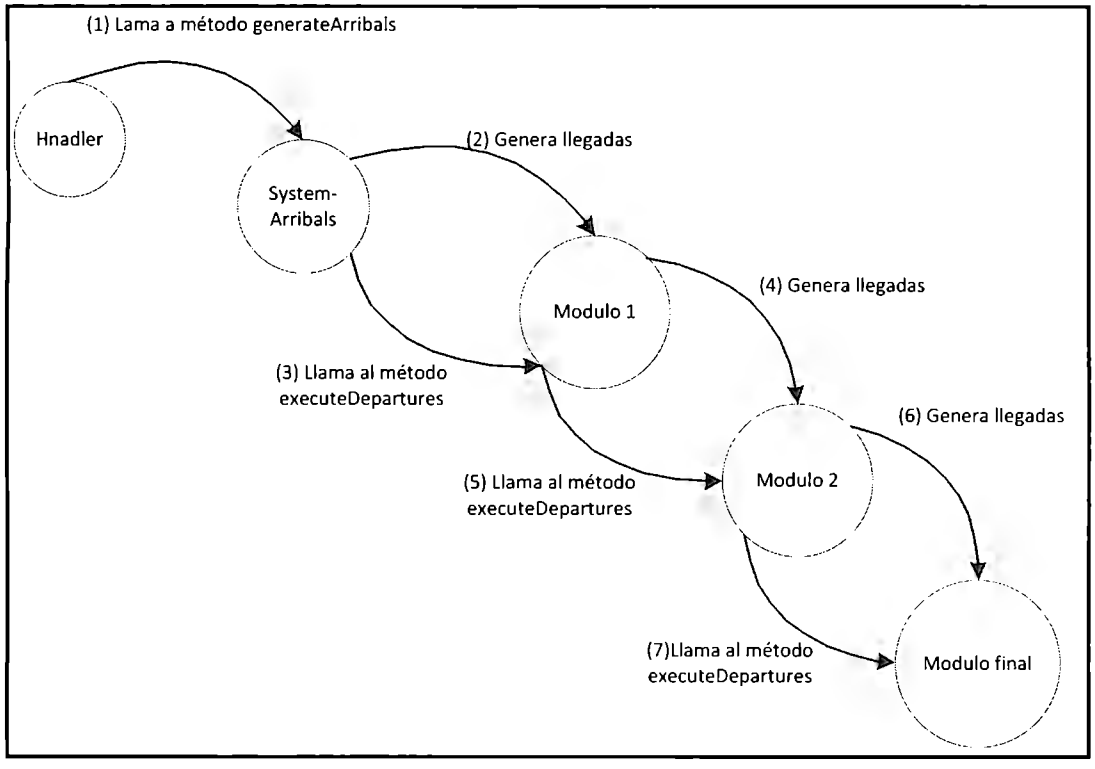


Figura 3. 23: Diagrama de la ejecución del simulador

Al llamarse al método de la clase **Handler** que inicia la ejecución de la simulación, éste utiliza una instancia de la clase **SystemArrivals** para generar llegadas al sistema, haciendo uso de su método **generateSystemArrivals**, el cual genera una lista de eventos de llegada en base a una distribución estadística determinada, a partir de la cual genera a su vez llegadas al primer módulo del sistema, una vez que haya generado todas las llegadas de artículos al primer módulo, le pedirá a éste, por medio del método **executeDepartures** que haga lo mismo con el siguiente módulo. El método **executeDepartures** genera llegadas al siguiente módulo en base a la lista de eventos de salida que tiene el módulo al que pertenece y después llama al método **executeDepartures** del siguiente método (en el caso de los módulos tipo "pull", la lista de eventos de partida estará vacía, por lo que prácticamente solo llamará al método **executeDepartures** del siguiente método). Este proceso se repetirá hasta llegar al último

módulo, el cual será instanciado mediante otro constructor y su proceso **executeDepartures** no hará lo mismo. La Figura 3. 23 ilustra la secuencia en la cual van a funcionar los diferentes módulos del simulador.

Una vez que el módulo final haya terminado de recibir piezas y almacenarlas, la clase Handler pedirá a la clase Schedule que ordene los eventos cronológicamente y que guarde un registro de estos. Al ser ordenada cronológicamente la lista de eventos se simula un funcionamiento concurrente de los diferentes módulos.

3.6. Diseño del Simulador como Sistema Distribuido Accesible desde Dispositivos Móviles

Ahora, se debe pensar en el simulador explicado anteriormente pero como elemento fundamental de un sistema distribuido accesible desde dispositivos móviles. En esta sección se describen la estructura, los componentes y el funcionamiento de dicho sistema de manera que se cumpla con los objetivos en cuanto a la accesibilidad, la representación gráfica de las simulaciones y el uso en equipos de trabajo.

3.6.1. Recursos y Técnicas Utilizadas para el Desarrollo del Simulador Móvil Distribuido

A continuación se da una descripción de los patrones y arquitecturas que se eligió seguir en el diseño de la aplicación como sistema distribuido, además de una explicación acerca de por qué se eligió seguirlos, para posteriormente describir el diseño del simulador móvil distribuido.

3.6.1.1. Selección del Tipo de Arquitectura a Seguir

Se siguió el patrón Modelo-Vista-Controlador (MVC por sus siglas en inglés) separando el desarrollo de la funcionalidad de la aplicación (parte del software encargada de ejecutar la simulación en base a las entradas generadas por el usuario) de la lógica de control de la aplicación y de la lógica utilizada para la presentación, situada en el dispositivo móvil, y situarla en el servidor. De esta manera la lógica funcional queda ajena a las limitaciones y aspectos

específicos a considerar para el desarrollo de aplicaciones para dispositivos móviles, los cuales van además a variar entre diferentes dispositivos. De modo que la lógica funcional es dejada casi intacta al implementarse la aplicación en dispositivos móviles.

Se utilizó la arquitectura basada en servicios principalmente debido a que:

- Permite agrupar código en base a un servicio completo.
- Permite ejecutar la lógica de funcional en un servidor de forma remota, evitando modificar el código para implementarlo en otros tipos de dispositivos (como dispositivos móviles).
- Permite del mismo modo hacer uso de las capacidades de cómputo de un servidor, evitando los posibles problemas de hacer uso de dispositivos con capacidades limitadas.
- Permite crear otros clientes y utilizar servicios complementarios (como pueden ser tutores inteligentes, aplicaciones de análisis u otras) en cualquier tipo de plataforma y con cualquier tipo de lenguaje.
- Los desarrolladores de las aplicaciones cliente y aplicaciones complementarias no necesitan conocer la implementación del simulador, sino únicamente los métodos expuestos por el servicio Web.

3.6.2. Descripción General de la Arquitectura del Sistema

Se describe de manera general la estructura y distribución de la aplicación completa, de la cual sus partes se describen en las siguientes secciones, exceptuando la lógica de funcional (el simulador), la cual fue descrita en la sección anterior.

La arquitectura utilizada se basa en el uso de un servicio Web encargado de manejar la interacción entre los usuarios pertenecientes a un grupo y entre dichos usuarios y el simulador, para esto los dispositivos de los usuarios utilizarán una aplicación cliente que utilice el servicio Web mencionado

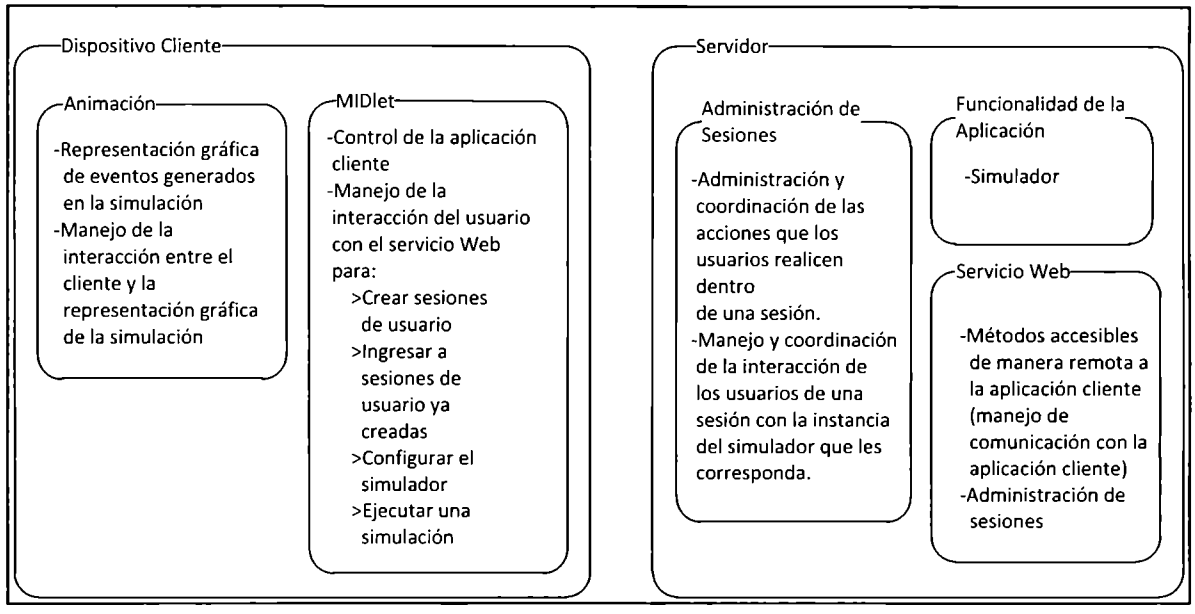


Figura 3. 24: Estructura del software de la aplicación

Como se mencionó anteriormente, se creó un servicio Web que utiliza la lógica funcional, localizada del lado del servidor, haciendo uso de los métodos públicos del simulador para configurarlo y ejecutar simulaciones, y que también maneja la administración de sesiones, actuando como controlador de la aplicación. Para esto se dividió la parte de la aplicación situada en el servidor en tres (ver la Figura 3. 24):

- **La funcionalidad de la aplicación (referente al simulador como tal):** va a contener toda la lógica del simulador como tal, exceptuando la representación gráfica de los eventos que se generen en la simulación.
- **el servicio web:** va a contener los métodos que serán accesibles a la aplicación cliente de forma remota, de modo que manejará la comunicación de la aplicación cliente con el software situado en el servidor. También contendrá toda la lógica que será utilizada para crear y administrar sesiones de grupos de usuarios, así como para ingresar a ellas. El servicio Web actúa como controlador de la aplicación y es el que usará la lógica de administración de sesiones para actualizar las sesiones de acuerdo a las acciones de los usuarios.
- **la lógica de administración de sesiones:** contiene las clases necesarias para administrar las sesiones de grupo y para permitir que los integrantes de dichas sesiones puedan

configurar y ejecutar de manera concurrente el simulador. Es mediante las clases que pertenecen a esta agrupación que el usuario tiene acceso al simulador.

La aplicación cliente va a manejar la interfaz de usuario (las pantallas que este verá y su interacción con la aplicación) y desplegará la animación de los eventos generados en las simulaciones que el usuario ejecute junto con sus compañeros. Esta aplicación va a estar dividida en dos partes principales (ver Figura 3. 24):

- **VisualMidlet:** Va a constar de una clase de tipo MIDlet que va a encargarse de proporcionar al usuario una interfaz mediante la cual pueda interactuar con sus compañeros de sesión y con el simulador, situado en el servidor.
- **Animación:** Va a encargarse de generar una representación gráfica al cliente de los eventos generados en la simulación, así como de manejar la interacción cliente-animación, de manera que pueda, por ejemplo, aumentar o disminuir la velocidad a la cual se despliegan los eventos en la simulación.

En base a lo anterior se puede observar que el cliente va a interactuar directamente con el servicio Web, el cual va a utilizar la lógica de administración de sesiones para crear y administrar sesiones de grupos de usuarios, y para permitir que dichos usuarios puedan compartir la misma instancia del simulador y ejecutarla de manera concurrente. En base a las peticiones de los usuarios el servicio Web va a crear sesiones de usuarios y a registrar usuarios en ellas, una vez dentro de la sesión, el usuario realizará acciones en la sesión, como ingresar o salir, seleccionar y configurar módulos del simulador, o escribir mensajes a los compañeros de sesión, esta última característica no se incluyó en la versión final debido al reducido tamaño de la pantalla de los dispositivos móviles, y a que se tiene que mostrar una cantidad considerable de información, como son la configuración que tienen los diferentes módulos y el estatus que tienen los diferentes usuarios presentes en la sesión. Ver Figura 3. 25.

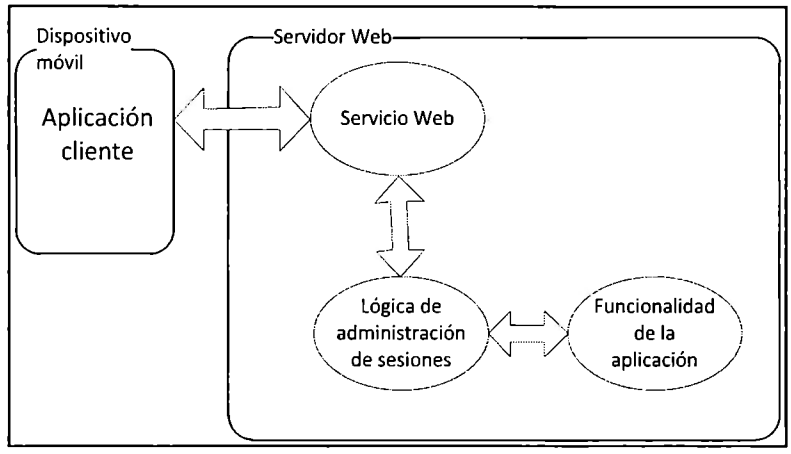


Figura 3. 25: Funcionamiento general de la aplicación

3.6.3. Diseño de la Aplicación del Lado del Servidor

Aquí se describe la estructura de la parte de la aplicación situada en el servidor. La Sección 3.6.3.1. (Paquete Session), Describe las clases que contiene el paquete sesión y la función de éstas, La sección 3.6.3.2. (Servicio Web) describe la clase contenida en el paquete del mismo nombre.

El código del lado del servidor va a estar dividido en tres paquetes (ver Figura 3. 26):

- **Session:** Contiene todas las clases necesarias para el control de sesiones.
- **Servicio Web:** Este paquete va a contener una sola clase, el Servicio Web, la cual va a contener todos los métodos disponibles para la aplicación del lado del cliente, y va a actuar como controlador de la aplicación.
- **Simulator:** Este paquete va a contener toda la lógica funcional del simulador.

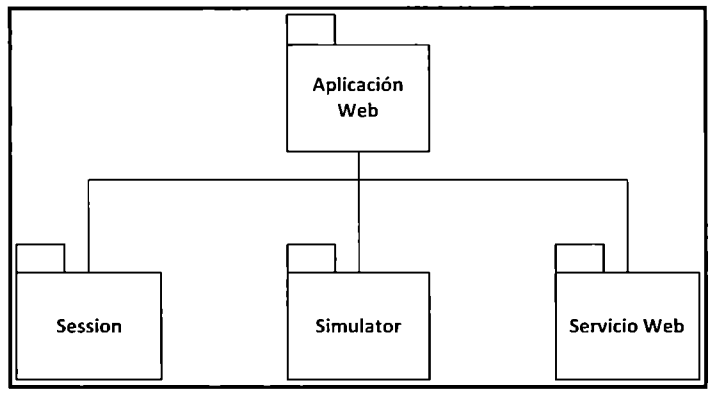


Figura 3. 26: Estructura de la aplicación servidor

3.6.3.1. Paquete Session

Este paquete contiene toda la lógica usada por el servicio Web para la administración de sesiones. La clase principal de este paquete es **Session**, por cada grupo, el servicio Web va a crear una instancia de esta clase que lo represente, y dará acceso a ella a todos los usuarios registrados en dicho grupo. Al servir el servico Web como intermediario entre la aplicación cliente y una instancia particular de la clase **Session**, muchos de sus métodos únicamente van a llamar a un método correspondiente de dicha instancia.

La clase **Session** lleva un registro de los siguientes aspectos:

- Cuáles compañeros se encuentran registrados en todo momento y cuál es su estatus (si se encuentran listos para ejecutar la simulación y qué módulo tienen seleccionado).
- Cuál es la configuración que el grupo ha dado a cada módulo de la línea de ensamble.
- Cuales módulos se encuentran disponibles para que los seleccione el usuario.

Para llevar a cabo la administración de sesiones la clase **Session** se auxilia de las siguientes clases (ver Figura 3. 27):

- **Module:** Se crea una instancia de ésta por cada módulo del simulador, esta clase va a representar un módulo de la línea de ensamble, y va a contener un conjunto de

parámetros que representen su configuración, además de un indicador que determine si está disponible o ha sido apartado por otro usuario.

- **Status:** Se crea una instancia de esta clase por cada usuario, esta lleva un registro de si el usuario está listo para ejecutar la simulación y que módulo tiene actualmente apartado.

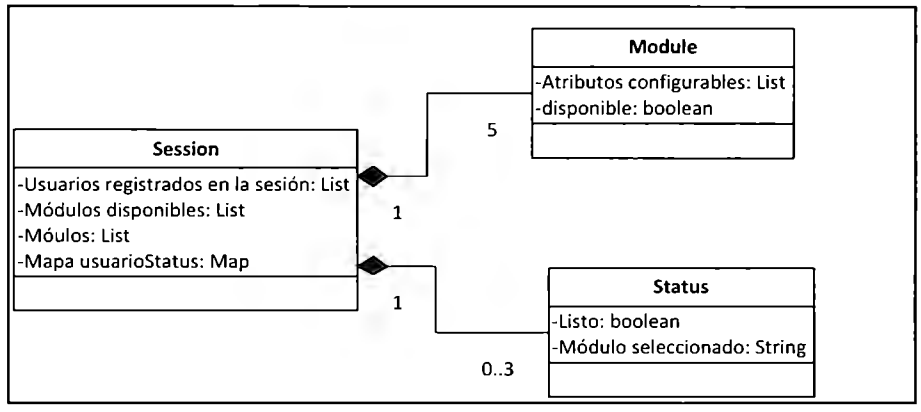


Figura 3. 27: Bosquejo del diagrama de clases parcial en torno a la clase Session

Para ejecutar la simulación, cuando los usuarios lo requieran, el servicio web llamará al método **StartSimulation** del objeto **Session** con el que se encuentran registrados dichos usuarios, el cual realizará las siguientes actividades (ver Figura 3. 28):

1. Verificará que todos los usuarios de la sesión están listos (en su status).
2. Sí todos los usuarios están listos creará una instancia de la clase principal del simulador (**Handler**).
3. Configurará el simulador de acuerdo a las configuraciones los objetos tipo Modulo que tiene enlistados, esto a través de la clase **Handler** creada.
4. Llamará al método **startSimulation** del objeto **Handler** para echar a andar la simulación.
5. Una vez que haya finalizado la simulación enviará la lista de eventos generados a cada una de las aplicaciones cliente.

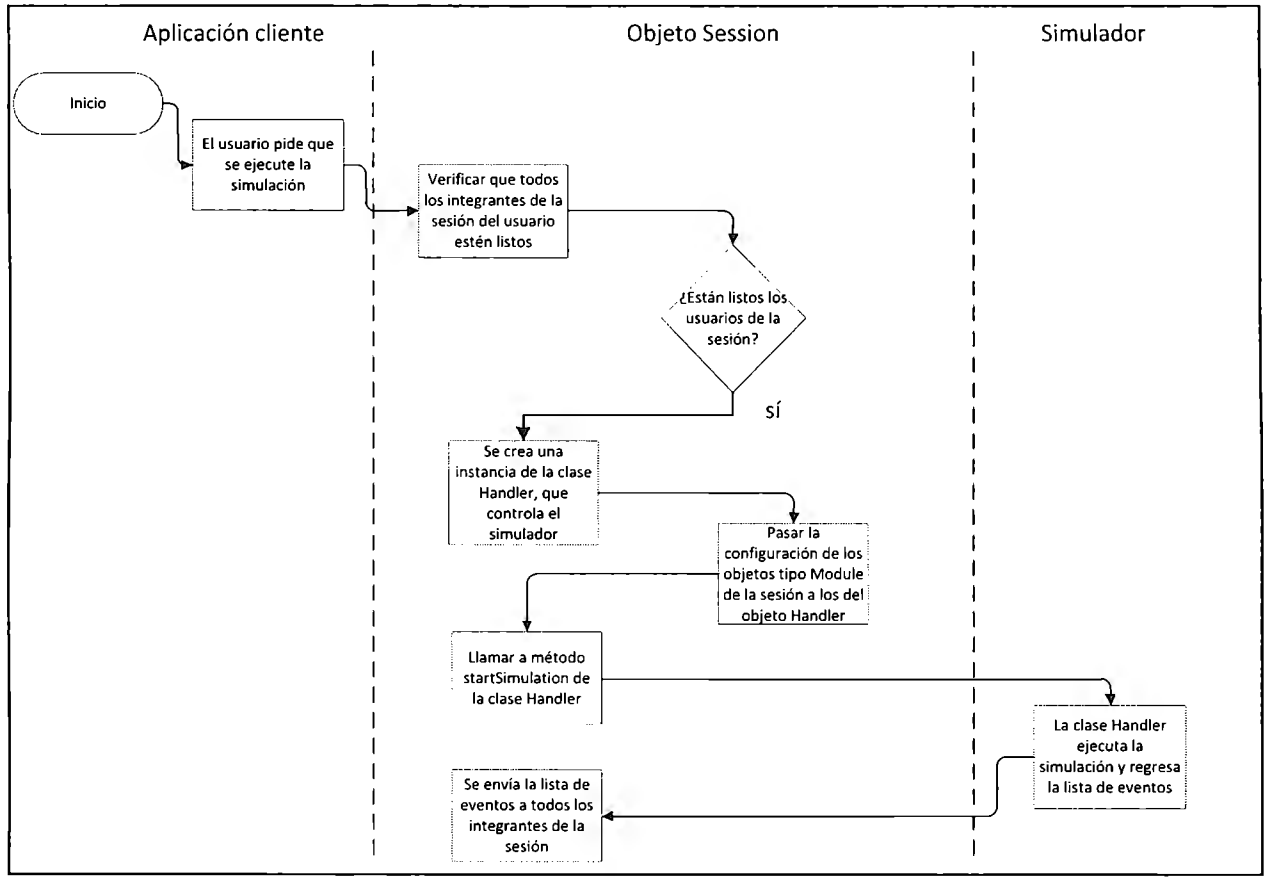


Figura 3. 28: Diagrama de flujo de la ejecución de simulaciones

Para enviar la lista de eventos a los usuarios lo que hace es (ver Figura 3. 29):

1. Aumentar en uno la variable **changesNum**, la cual lleva el registro de la cantidad de cambios que ha habido en la sesión, la aplicación cliente revisará constantemente esta variable.
2. Cambiar el valor de la variable booleana de sesión **SimulationStarted** de falso a verdadero, la aplicación cliente revisará esta variable al enterarse de que ha habido un nuevo cambio en la sesión, por medio de la variable **changesNum**)
3. Cuando el cliente vea que ha iniciado la simulación éste pedirá, por medio del servicio web la lista de eventos generados en la simulación.

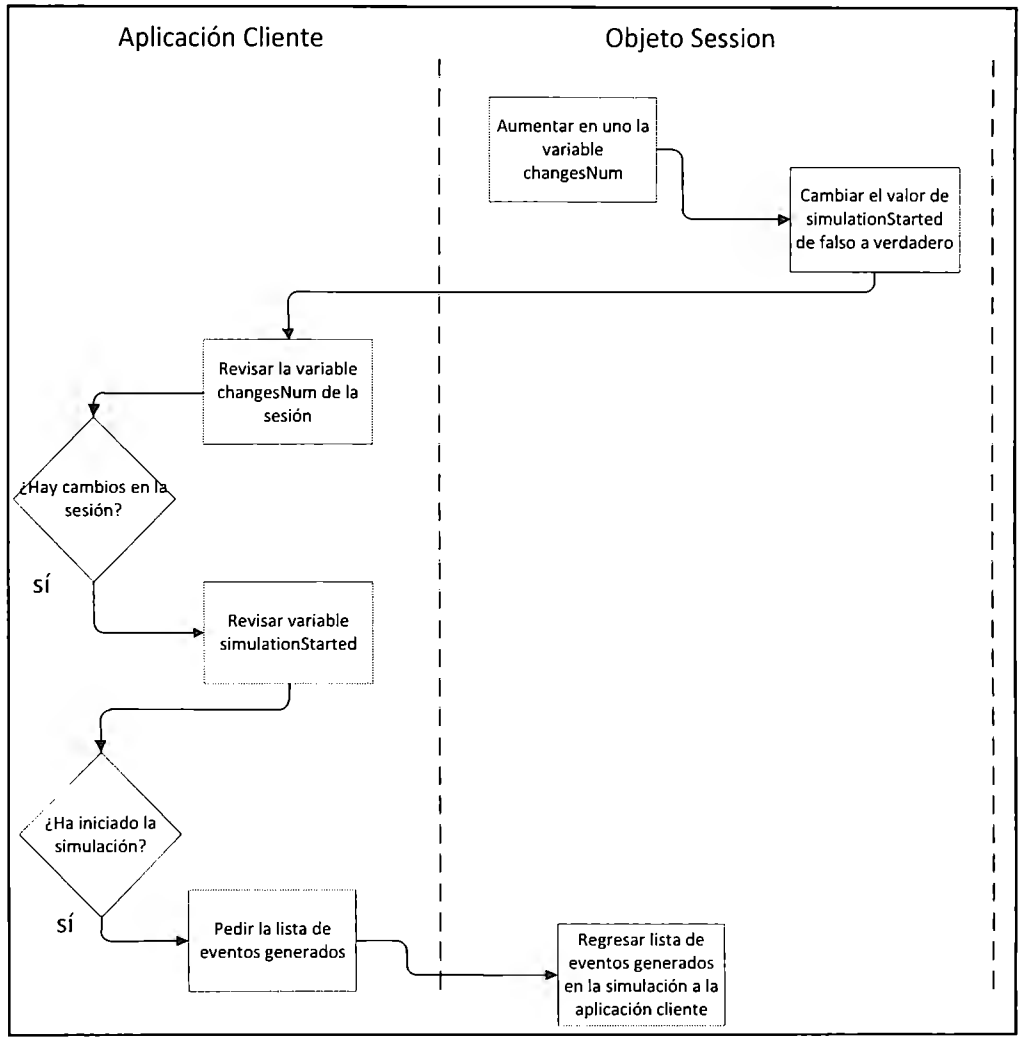


Figura 3. 29: Diagrama de flujo del proceso de envío de la lista de eventos al cliente

3.6.3.2. Servicio Web

El servicio web sirve de enlace entre el cliente remoto y, la lógica del simulador y de administración de sesiones. Dicho servicio web expone las interfaces públicas de la lógica de sesión utilizadas por las aplicaciones cliente. Este servicio web va a publicar también todos los métodos necesarios para que el cliente pueda crear sesiones y, registrarse e interactuar con ellas. Así mismo llevará un registro de qué sesiones existen y qué usuarios están registrados en cada una de ellas.

Las sesiones serán representadas por objetos de tipo Sesión, de las cuales el servicio Web tendrá una lista y un mapa para asignarles usuarios. El servicio Web, al servir de enlace entre el cliente y el objeto de tipo **Session** que represente el grupo al que pertenece, muchos de los métodos que tiene publicados solamente llaman a un método de la clase **Session**.

El cliente podrá crear una sesión de la siguiente manera (ver Figura 3. 30):

a) El cliente llama al método crear sesión (**createSession**) del servicio web.
b) El servicio web, si aún hay lugar para un grupo nuevo, crea una instancia de la clase **Session** y la almacena en una lista de objetos de ese tipo.
c) El cliente llama al método ingresar a sesión (**joinSession**) del servicio Web y este lo asigna a un objeto de tipo **Session** (correspondiente al grupo que ha acabado de crear), por medio de un mapa. Del mismo modo se le asigna un objeto de tipo estatus al cliente. La clase Status va únicamente a tener variables que indiquen qué modulo ha seleccionado el cliente y si este se encuentra listo para que se ejecute la simulación.

El cliente va a interactuar con el objeto sesión que le corresponda de la siguiente manera (ver Figura 3. 31):

a) El cliente llama a un método publicado por el servicio web.
b) El método llamado por el servicio Web llama al método correspondiente del objeto sesión.
c) El método de la sesión llamado manda una respuesta al servicio web.
d) El método llamado del servicio Web manda la respuesta recibida de parte de la sesión, a la aplicación cliente.

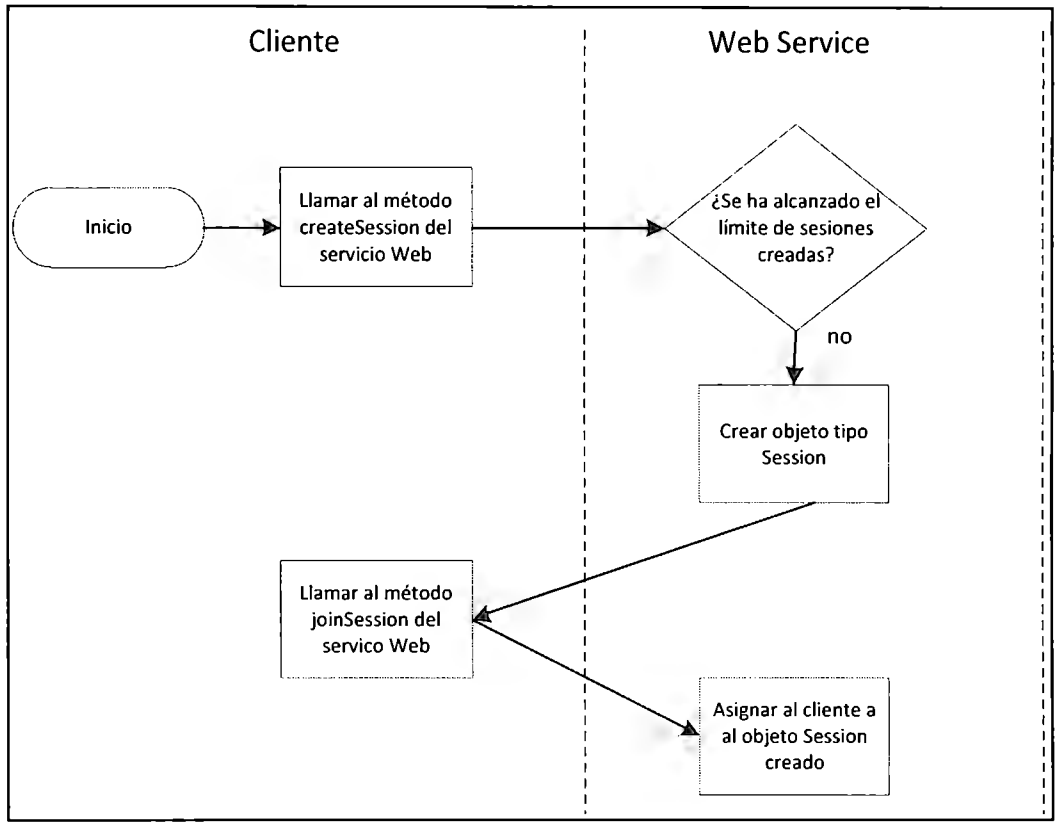


Figura 3. 30: Diagrama de flujo de la creación de sesiones

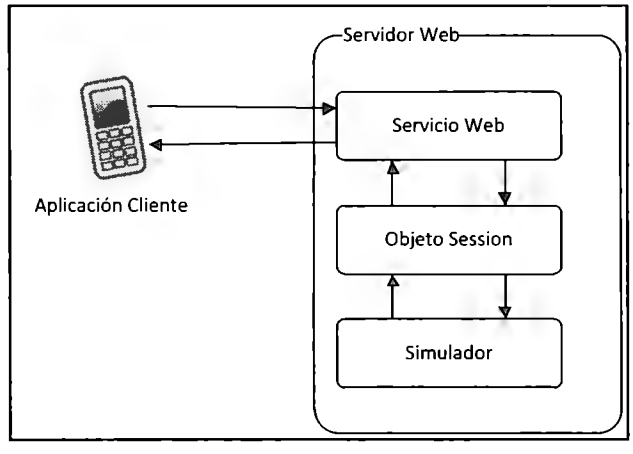


Figura 3. 31: Interacción del cliente con la sesión

Para hacer un uso eficiente de los recursos de red, el servicio Web pone a disposición de la aplicación cliente el método **getChangesNum**, dicho método regresará al usuario un valor entero

que representará la cantidad de cabios realizados en la sesión a la que pertenece (**changesNum**), de manera que, para estar al pendiente de los cambios realizados en su sesión, es decir, si algún compañero de sesión apartó o configuro un módulo, o si entró a la sesión o la abandonó, entre otros, se mantendrá pidiendo al servicio Web únicamente la cantidad de cambios realizados en esta, para solo actualizar los parámetros que tiene de dicha sesión, como son la lista de usuarios y la lista de módulos, si ha habido un cambio en ella. De este modo se reduce la cantidad de información enviada y recibida a través de la red inalámbrica, por lo que se reducen los posibles problemas en la velocidad de la aplicación relacionados con la poca confiabilidad de las redes inalámbricas.

3.6.4. Diseño de la Aplicación Cliente (Vista)

En esta sección se describen la estructura y el funcionamiento de la parte de la aplicación situada en el dispositivo móvil del cliente. La sección 3.6.4.1. (Selección del Tipo de Aplicación Móvil a Desarrollar) describe el proceso de selección del tipo de aplicación móvil a desarrollar, por su parte, la sección 3.6.4.2. (Descripción del Diseño de la Aplicación cliente) describe la estructura y el funcionamiento de la aplicación cliente (la parte de la aplicación que va a estar situada en el dispositivo móvil del cliente).

3.6.4.1. Selección del Tipo de Aplicación Móvil a Desarrollar

Se decidió hacer uso de Java Micro Edition para el desarrollo de la aplicación cliente ya que permite hacer uso de las capacidades de Java para el manejo de varios hilos de ejecución, necesarios para la creación de aplicaciones multiusuario que requieran interacción en tiempo real y para el despliegue de animaciones. Además se dispone de sus herramientas para la creación de video juegos, que facilitan considerablemente la creación de dicho tipo de aplicaciones, las cuales pueden ser utilizadas para generar representaciones dinámicas de simulaciones en dispositivos móviles, se dispone también de herramientas para el control de botones del dispositivo, la creación de imágenes dinámicas y el control de los mismos *threads* (hilos de ejecución).

A pesar de que los *widgets* basados en tecnología Web permiten hacer uso de AJAX para mantener una comunicación constante con una aplicación en un servidor, así como hacer uso de servicios Web, no cuentan con paquetes similares a los de MIDP para desarrollo de videojuegos, los cuales facilitan considerablemente la creación de animaciones interactivas útiles para representar simulaciones. Los paquetes mencionados ayudan en la creación de secuencias de imágenes y su despliegue en pantalla, la manipulación de hilos de ejecución y la interacción con el usuario por medio del teclado del dispositivo, por lo que representan una buena ventaja por parte del uso de Java ME como herramienta para la creación de la aplicación cliente. Debido a lo anterior y al mayor conocimiento que se tiene del desarrollo de aplicaciones móviles en Java, se eligió desarrollar la aplicación en base a MIDP.

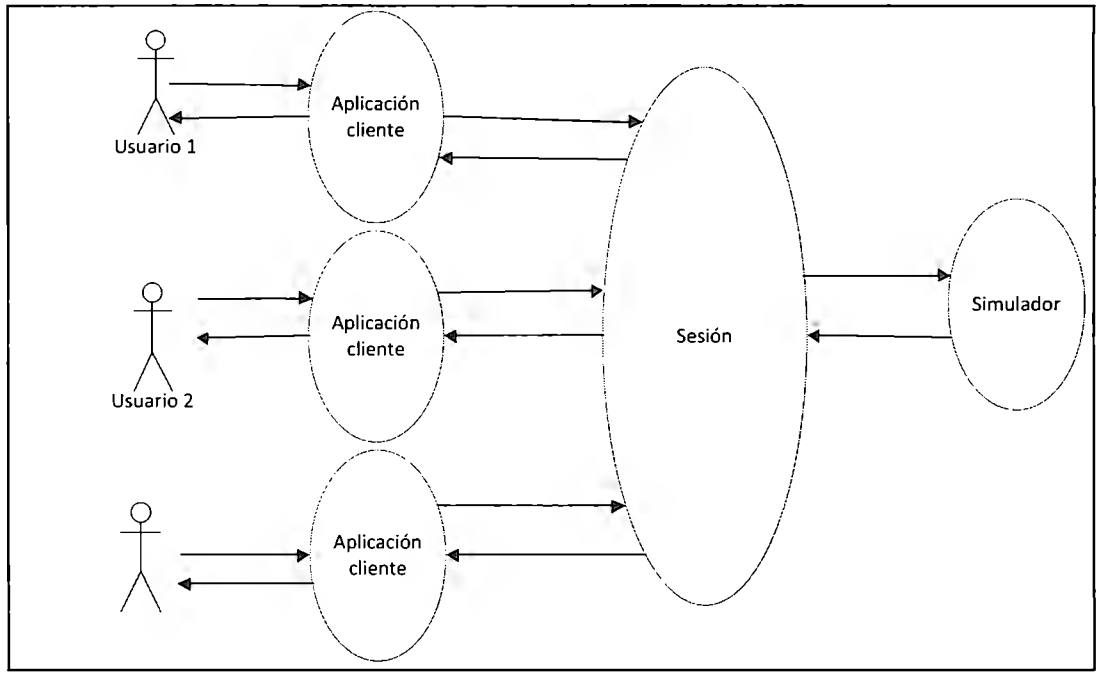


Figura 3. 32: Uso de la aplicación cliente

El componente de la aplicación situado en el dispositivo móvil del usuario se va a encargar de dar al usuario una interfaz mediante la cual este pueda, crear sesiones o ingresar a ellas, interactuar con los usuarios pertenecientes a su grupo de trabajo y con el simulador (para configurar sus parámetros y ejecutar la simulación), y observar la presentación animada en tiempo real de la simulación ejecutada (ver Figura 3. 32).

3.6.4.2. Descripción del Diseño de la aplicación Cliente

Tanto la interfaz como la presentación de los resultados fueron realizadas usando exclusivamente APIs pertenecientes a MIDP 2, sin usar APIs exclusivas para BlackBerry (dispositivo en el que se piensa utilizar la aplicación[24]) para darle mayor portabilidad, que eventualmente pudiera ser requerida.

La suite del MIDLet[25] se compone de los siguientes elementos principales (ver Figura 3. 33):

- VisualMidlet: Este componente va a encargarse de desplegar las diferentes pantallas que el cliente va a ver, así como de manejar la interacción con el usuario por medio del despliegue en pantalla de botones.
- AuxialThread: Este componente se va a encargar de actualizar al VisualMidlet en relación con las sesiones de grupo (que acciones han realizado los compañeros en dicha sesión). Funcionando como una especie de "*listener*" hacia afuera del cliente y se le va a asignar un hilo de ejecución.
- ClsCanvas: Este componente se va a encargar de desplegar una animación de los diferentes eventos generados en la simulación, a partir de una lista de eventos generaos.

[24] Se decidió realizar el proyecto para BlackBerry ya que el Tecnológico de Monterrey cuenta con un programa en el cual se le brinda a los alumnos de nuevo ingreso un dispositivo de este tipo, además, es el dispositivo con el que se cuenta en el laboratorio para realizar pruebas de la aplicación.

[25] En la sección 2.8.3.1 se explica qué es una *suite* de MIDlet.

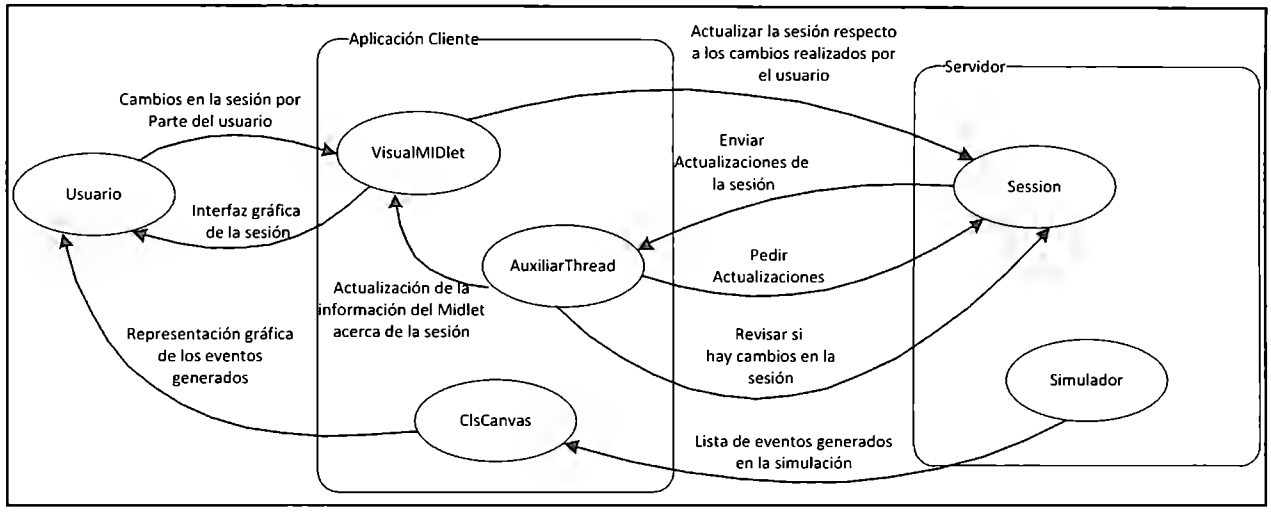


Figura 3. 33: Estructura y funcionamiento del software en el dispositivo del usuario

A continuación se describe el funcionamiento del Componente VisualMidlet, que es ilustrado mediante el diagrama de estado de la Figura 3. 34:

1) El usuario ingresa al menú de grupos, donde encontrará un listado de las diferentes sesiones de grupo activas, así como la opción de crear una sesión de grupo nueva.
2) Si el usuario decide ingresar a un grupo ya existente será dirigido a la pantalla de sesión de grupo, si decide crear un grupo nuevo será enviado primero a la pantalla de creación de grupos donde se le preguntará el nombre del grupo a crear. En la pantalla de sesión podrá ver y manipular la configuración que se le dará al simulador, así como observar las acciones de sus compañeros de sesión.
3) Una vez en la pantalla de sesión el usuario podrá escoger un módulo de la línea de ensamble simulada, el cual podrá configurar al escoger la opción configurar (configure) del menú de comandos (cada módulo solo podrá ser seleccionado por un usuario a la vez).
4) Cuando el usuario quiera iniciar la ejecución de la simulación, este elegirá la opción de "iniciar simulación" del menú de comandos, si todos los integrantes de su sesión se encuentran listos en su estatus se ejecutará la simulación para posteriormente enviarle a cada usuario la lista de eventos generada.

5) Una vez que la aplicación cliente tenga la lista de eventos generada le delegará el control de la pantalla al componente encargado de la animación de los eventos.
6) Cuando el cliente quiera volver al menú de grupos solo tendrá que apretar la tecla "*fire*" del dispositivo móvil.

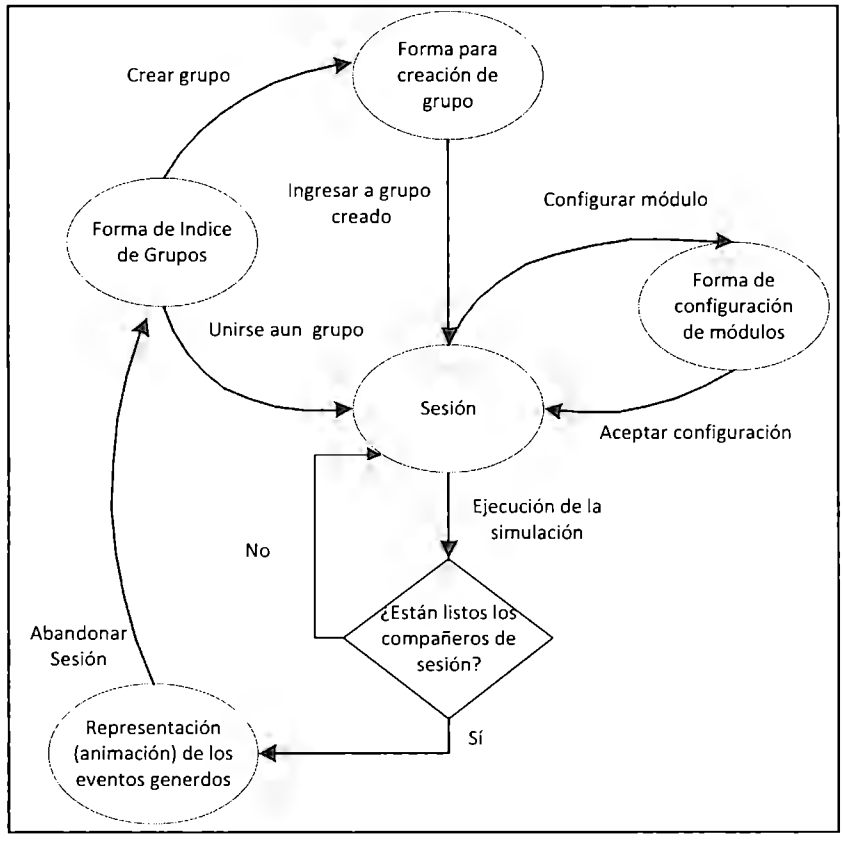


Figura 3. 34: Diagrama de estado del elemento VisualMidlet del cliente

A continuación se describe el funcionamiento del componente AuxiliarThread de la aplicación cliente (ilustrado mediante el diagrama de flujo de la Figura 3. 35):

1) Revisa, mediante el servicio Web, si ha habido algún cambio en la sesión en la que el usuario está registrado.
2) De existir algún cambio revisa si ha comenzado la ejecución de la simulación (si algún usuario seleccionó la opción de "iniciar la simulación" y todos los clientes estaban listos en su estatus).
3) Si no ha comenzado la ejecución de la simulación procede a descargar la lista de status de usuarios y la lista de status de módulos para actualizar su información y la del

VisualMIDlet acerca de la sesión, para posteriormente seguir revisando si ha habido más cambios en su sesión.

4) De haber comenzado la ejecución de la simulación procede a esperar a que esté disponible la lista de eventos generados en la simulación para descargarla y activar la animación de eventos.

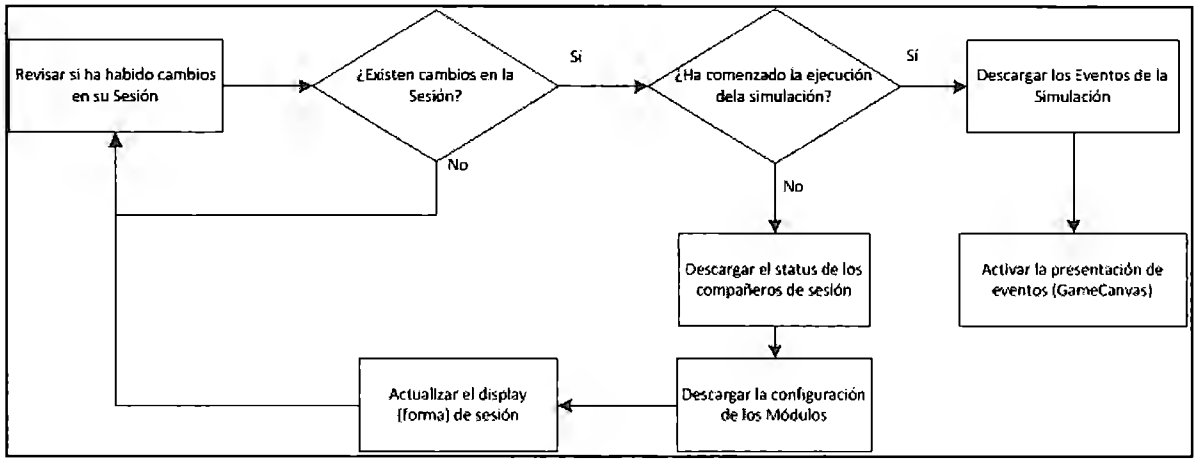


Figura 3. 35: Diagrama de flujo para el elemento ThreadAuxiliar de la aplicación cliente

A continuación se enlista el proceso para el despliegue de eventos resultantes de la simulación (ilustrado mediante el diagrama de flujo en la Figura 3. 36):

1) Tomar de la lista de eventos generada en la ejecución del simulador el primer evento enlistado.
2) Actualizar las colas de los módulos y la ocupación de los almacenes de acuerdo al tipo de evento que se haya tomado de la lista (se utilizan objetos que representen las colas y la ocupación de los almacenes, estos solo servirán para crear representaciones y no implicarán lógica alguna).
3) Imprimir la representación delas colas y la ocupación de los almacenes en pantalla.

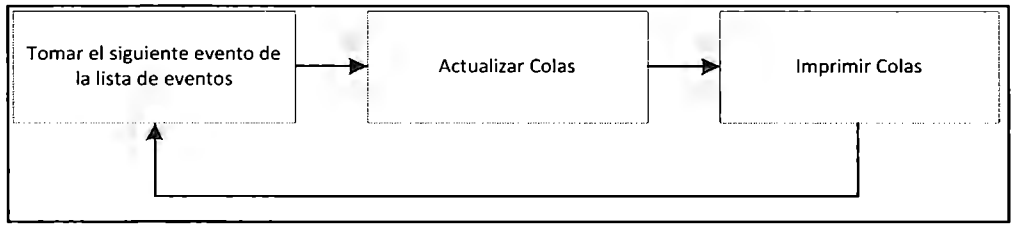


Figura 3. 36: Diagrama de lujo para el elemento clCanvas de la aplicación cliente

Capítulo 4

Implementación

En este capítulo se detalla la programación de la aplicación y su implementación en el servidor y en los dispositivos cliente, la Figura 4. 1 muestra la estructura general de las clases de la aplicación. En dicha figura se puede observar el papel de cada componente en el funcionamiento de la aplicación, el elemento VisualMidlet sirviendo de interfaz al usuario para interactuar con el objeto sesión (**Session**) al que pertenece, usando para ello los métodos publicados por el servicio Web. El objeto **Session** configura y ejecuta el simulador de acuerdo a las peticiones hechas por el usuario, enviando de regreso la información de la sesión y los eventos resultantes de la simulación de regreso a la aplicación cliente, para que ésta genere una animación.

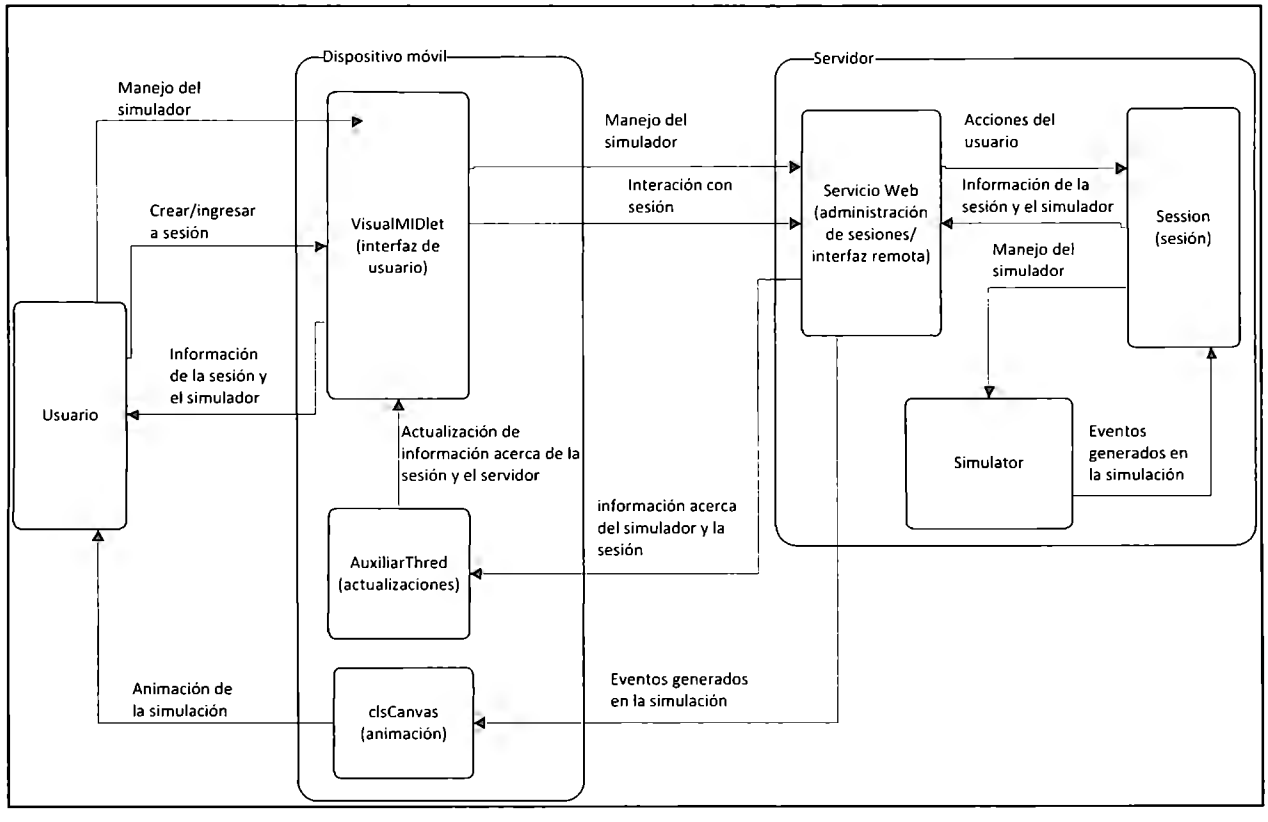


Figura 4. 1: Estructura general de las clases de la aplicación

4.1. Implementación Funcional.

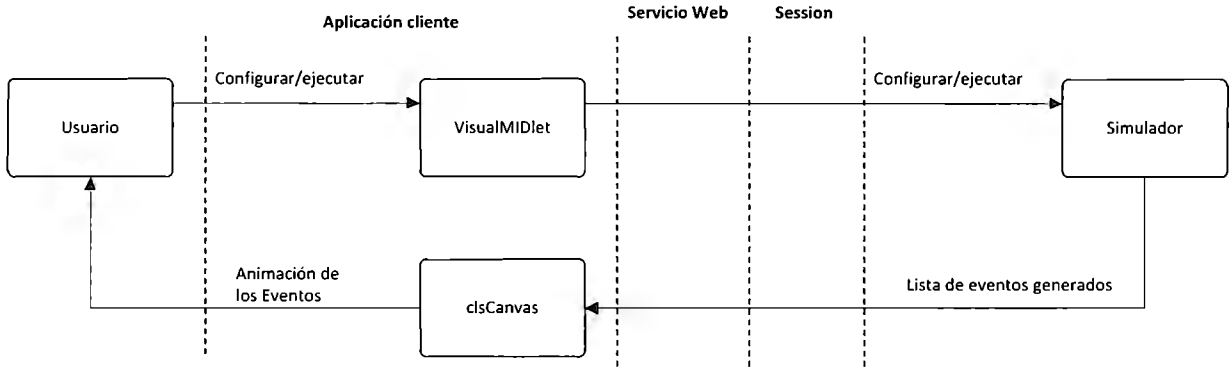


Figura 4. 2: Esquema general de la aplicación en torno al uso de la lógica funcional

Recordemos que la lógica funcional se refiere a todo lo relacionado con la función central de la aplicación, que en este caso es el simulador, y que se decidió, como se menciona en la sección 3.5.1, seguir el patrón de orientado a eventos para la modelación del sistema a modelar, así como utilizar únicamente las librerías estándar de Java, como se mencionó en la sección 3.5.2, para la implementación en código.

A continuación se explica a detalle la programación del simulador de la línea de ensamble del laboratorio de Ingeniería Industrial del Tecnológico de Monterrey, cómo se programó a detalle cada una de las clases que componen el simulador y qué clase se utilizó para introducir una representación de cada uno de los módulos de la celda de ensamble en el código del simulador, además de explicar cómo se introdujeron las características específicas de cada uno de los módulos en dicho código.

La Figura 4. 2 muestra el esquema general de la aplicación en torno al uso que se le dará a la lógica funcional, en la cual las configuraciones y ejecuciones hechas por el usuario por medio de su dispositivo móvil son llevadas al simulador a través del servició Web y la sesión. Regresando

el simulador una lista de eventos, a partir de la cual la aplicación cliente realizará una representación gráfica por medio del componente clsCanvas.

4.1.1. Programación de los Aspectos Generales del Simulador

En esta sección se describe a detalle la programación de los aspectos del simulador presentes en cualquier línea de ensamble de este tipo, es decir todas las clases necesarias para programar el simulador excepto las subclases e instancias específicas de la celda de manufactura a simular.

Se creó una clase llamada **Module** a partir de la cual se generarían instancias que representen los diferentes módulos a modelar del sistema, además de algunas otras clases principales para la ejecución de las simulaciones, las cuales se describen más adelante.

El simulador va a depender para su funcionamiento de una clase llamada **Event**, la cual representa dos tipos de eventos, salidas y entradas a los módulos, por medio de tres variables:

1. El módulo en el que se dio dicho evento
2. El tiempo en que se dio dicho evento
3. El artículo o *pallet* implicado (representado por medio de la clase DynamicEntity)

La clase **DynamicEntity** va a ser abstracta y tendrá un conjunto de variables que la van a describir que son: el nombre de la entidad, el color (en el caso de los *pallets* el color se va a referir al color de las piezas que contienen, existiendo la posibilidad de que sea mixto). De la clase **DynamicEntity** van a heredar las clases **Item** (que va a representar artículos individuales) y **pallet** (que representará agrupaciones de **Items**). La Figura 4. 3 muestra el diagrama de clases a partir de la clase **DynamicEntity**.

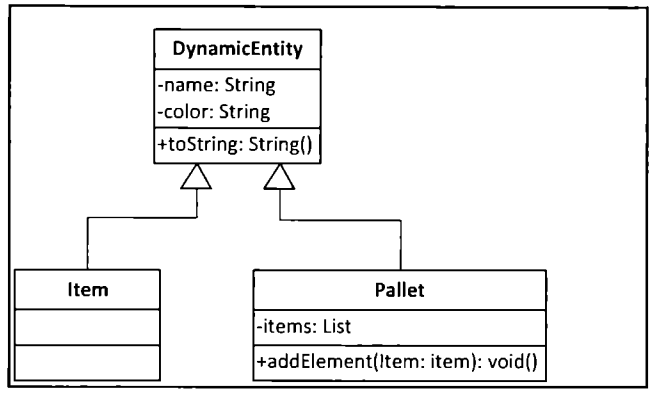


Figura 4. 3: Diagrama de clases a partir de la clase DynamicEnity

Otra clase sobre la cual va a girar el funcionamiento del simulador es la clase **Schedule**, esta clase va a contener una lista de eventos, a la cual van a estar agregando elementos los diferentes módulos del sistema cada vez que estos creen una instancia de tipo evento. Además va a contener un método que va a ordenar la lista de eventos en orden cronológico, de los cuales después va a imprimir una representación en un archivo de texto para crear un registro y, posteriormente generar una lista de eventos codificada la cual va a regresar a la clase **handler** para que esta la ponga a disposición del código encargado de crear la representación. Dada la relativa simplicidad del sistema se optó por solamente colocar una lista de eventos codificada en un **String** y crear una clase dedicada a la generación de eventos a partir de dicho **string** en la aplicación o el componente de código encargado de la generación de la representación visual de la simulación. El código 4.1 muestra un pseudocódigo de la clase **Schedule**.

Código 4.1. Pseudocódigo parcial de la clase Schedule.

```
Class Schedule{
    List eventsList;

    Void addEvent(Event event){
        eventsList.add(event)
        /*Las demás clases llamarán a este método para registrar  sus eventos
        Locales en una lista global de eventos*/
```

```
        }

        generateEvents(){
            Collections.sort(eventsList);
            //Se ordena cronológicamente la lista global de eventos
            /* Se genera el archivo de texto en el que se van a representar los
            Eventos generados y la lista de eventos codificada a regresar a la clase
            Handler */
        }
    }
```

La clase **Handler** se va a encargar de generar todas las instancias de la clase module que va a utilizar el simulador, y va a implementar métodos para la configuración de estos y para iniciar la ejecución de la simulación. De manera que, al agregarse el código del resto de la aplicación, solo se tenga que generar una instancia de la clase **Handler** y llamar algunos de sus métodos disponibles para hacer funcionar al simulador. La Figura 4. 4 muestra un diagrama de clases parcial en torno a la clase **Handler**.

Para llevar a cabo lo anterior la clase **Handler** hace uso de la clase **SystemArrivals**, la cual va generar una lista de eventos de llegada a partir de la clase **Distribution**, la cual va a generar una lista de tiempos de llegada en base a una distribución estadística determinada.

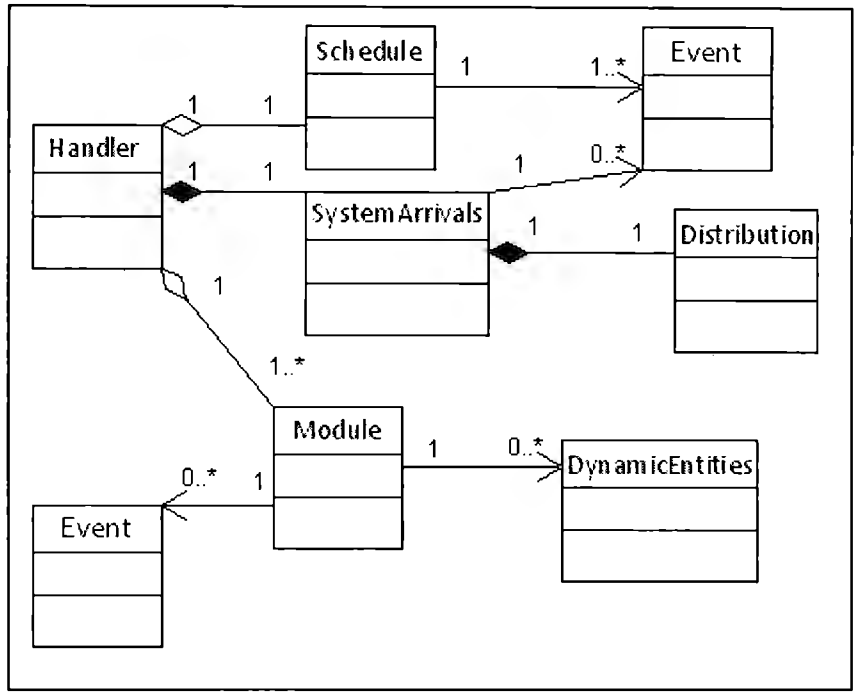


Figura 4. 4: Diagrama de clases parcial en torno a la clase Handler

Los principales métodos de la clase **Handler** se explican a continuación:

- **Método create:** Este método crea todos los objetos tipo **Schedule**, los objetos tipo **Module** y el objeto tipo **SystemArrivals** que van a ser utilizados por el simulador.
- **Métodos tipo setter:** Este conjunto de métodos va a permitir a código de la aplicación externo al simulador configurar parámetros de los objetos tipo **Module** que se van a utilizar.
- **Método startSimulation:** Este método inicia la ejecución del simulador, pidiendo al objeto tipo **SystemArrivals** que genere las llegadas de piezas al objeto **Módulo1**, para que inicie la ejecución de la simulación, una vez que se haya ejecutado la simulación por completo pedirá al objeto **Schedule** que ordené su lista de eventos en orden cronológico.

4.1.2. Programación en el Simulador de los Aspectos Específicos de la Celda de Ensamble.

En esta sección se explica qué clase se utiliza para modelar qué módulo de la celda de ensamble a simular, cómo se extiende dicha clase para representar aspectos particulares de dicho módulo, de ser el caso, y cómo se van a generar las instancias de dichas clases en el simulador.

Para hacer uso de polimorfismo, y por lo tanto reutilización de código, se creó una clase abstracta llamada **Module**, la cual tiene los métodos comunes para todos los módulos: llegada y salida, y de la cual heredan las clases tipo **PushModule** (que es modelada como módulo de tipo *push*) y tipo **PullModule** (modelada como módulo de tipo *pull*), las cuales utilizan diferentes implementaciones de ambos métodos. La Figura 4. 5 muestra el diagrama de clases a partir de la clase Module.

La clase tipo **PushModule** implementa la función de llegada calculando todas las salidas que ocurran antes de cada llegada, quitando las piezas que salen de una lista donde se registran los artículos que están presentes en el módulo en cuestión y creando eventos de tipo salida, los cuales se van a registrar en la lista ***departureList***. El módulo anterior, una vez que haya enviado la última pieza al módulo en cuestión, llamará al método **executeDepartures** (ejecutar Salidas) de este último para que envíe los artículos involucrados en cada evento salida registrado en la lista **departureList** al siguiente módulo. Colocando dichos artículos y sus tiempos de partida como argumentos de la función llegada del siguiente módulo y llamando la ejecución de dicha función. El siguiente módulo llamara a su vez, eventualmente, al método ejecutar salidas del módulo que le sigue. El código 4.2 muestra un pseudocódigo parcial de la clase **PushModule**.

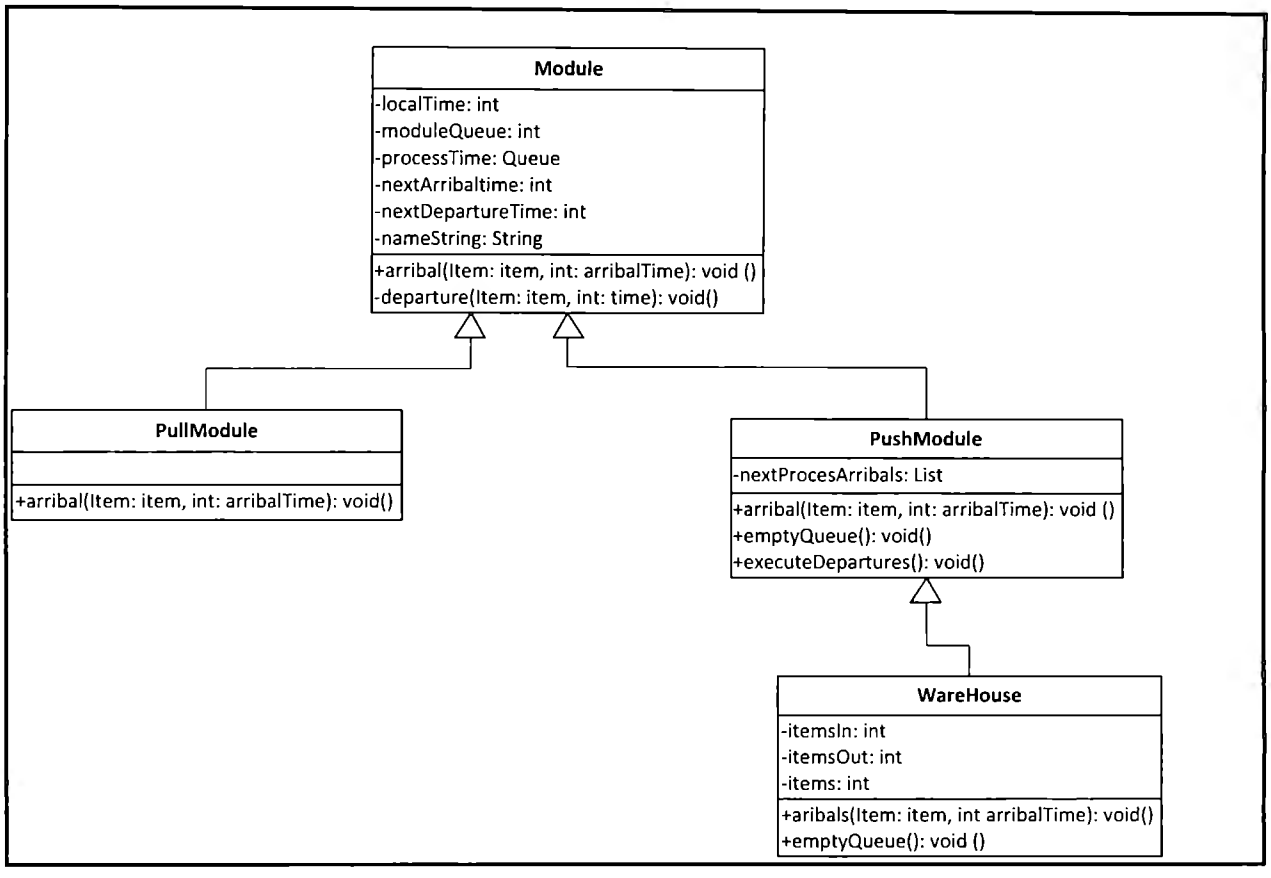


Figura 4. 5: Diagrama de clases a partir de la clase Module

Código 4.2. Pseudocódigo parcial de la clase PushModule.

```
Class PushModule{
    List departureList;
    void arrival(Item pieza, int tiempo){
        /* Se calcula el tiempo de salida de la pieza que llega, se actualizan
        las colas y se registran al mismo tiempo los eventos de llegada y de
        salida, tanto local como globalmente */
        Event arrivalEvent=new ArrivalEvent( pieza, tiempo);
        schedule.addEvent(arrivalEvent);
        Event deparureEvent=new DepartureEvent(pieza, tiempoDeSalida);
        Schedule.addEvent(departureEvent);
```

```
            departureList.add(departureEvent);
        }
        executeDepartues(){
            /*se ejecutarán llegadas al siguiente módulo de acuerdo con todos los eventos
             de salida registrados localmente*/
            Iterator iterator=departureList.iterator();
            while(iterator.hasNext){
                Event event=iterator.next();
                nextModule.arrival(event.piece, event.time);
            }
        }
    }
```

De la clase **ModuloPush** hereda la clase Almacén, la cual va a sobrescribir el método de llegada para únicamente almacenar los artículos y determinar las salidas en base a una condición en particular. Para ejecutar las salidas posteriores a la última llegada en todos los módulos tipo *Push*, se ejecutará una función que sacará cualquier artículo que haya quedado en la cola esperando a ser procesado y lo enviará al siguiente módulo. El código 4.3 muestra el pseudocódigo parcial de la clase **WareHouse** (almacén).

Código 4.3. Pseudocódigo parcial de la clase **WareHouse**.

```
Class WareHouse{
    Module nextMdule;
    List departureList;
    void arrival(Item piece, int time){
        /* Se calcula el tiempo y el lugar en el que la pieza que llega es almacenada
        y se verifican las condiciones para que se ejecuten salidas de piezas del
        almacén */
        if(departureCondition==true){
```

```
                scheduleDepartures();
            }
        }
        void scheduleDepartures(){
            /* Se calcula en un loop qué piezas van a salir y en qué tiempo y se
             Registran dichas salidas en la lista "listaDeSalidas", adicionalmente se
             registran los eventos de entrada y de salida en la lista de eventos global */
        }
        Void executeDepartures(){
            /* Se ejecutan todas lasllegadas al siguiente módulo de acuerdo a las salidas
            registradas en la lista "departureList" en orden cronológico */
        }
    }
```

La clase tipo **PullModule**, al igual que la clase de tipo **PushModule**, cada vez que es llamado su método de llegada, revisa qué salidas se dan antes de que llegue la pieza colocada en el argumento de dicho método, solo que, a diferencia de los módulos de tipo push, en lugar de colocar las salidas en una lista las va a sacar directamente invocando la función de llegada del siguiente módulo, la cual va regresar una variable de tipo entero llamada **endOfPprocess**, con el propósito de que las siguientes salidas se reprogramen de acuerdo al tiempo en que se acabó de procesar la última pieza en el siguiente módulo. El código 4.4 muestra un pseudocódigo parcial de la clase **PullModule**.

Código 4.4. Pseudocódigo parcial de la clase **PullModule**.

```
Class PullModule{
    Module nextModule;
    void arrival(Item piece, int arrivalTime){
        */se revisan los tiempos de salida de las piezas que se encontraban en el módulo
         al momento en que llegó la pieza*/
```

```
        while(nextDepartureTime<arrivalTime){
            /* nextDepartureTime es el tiempo en que se calcula que saldrá la siguiente
            pieza del módulo*/
             nextDepartureItem=ItemsQueue.getFirst();
             excuteDeparture(nextDepartureTime, nextDepartureItem);
        }
     /* Se almacena la pieza que llegó en la cola del módulo y se calcula el tiempo de la
     Siguiente salida */
   }
   void executeDeparture(int tiempoDeSalida, Item pieza){
      nextModule.arrival(timpoDeSalida, pieza);
   }
}
```

Se identificó qué módulos son de tipo *push* (que envían todos los artículos una vez que los haya procesado, sin importar que los manden a colas de espera) y los que son de tipo *pull* (que esperan a que el siguiente módulo esté desocupado para mandar la próxima pieza). Los módulos 1, 3, 4 y 5 se clasificaron como módulos tipo *push,* y el módulo 2 como de tipo *pull.*

Para cada módulo se debió identificar el tiempo que tarda en procesar cada pieza y el tiempo que toma a cada pieza o pallet desplazarse desde el módulo anterior hasta él (ya sea por medio de bandas, o cualquier método de transporte), pues este tiempo será sumado al tiempo con el que el módulo anterior llame a su método de llegada.

Del mismo modo se identificó qué módulos son de tipo **Wehouse** (que almacenan las piezas hasta que se dé una condición establecida), y cuáles son de tipo **Palletizer** (que reciben piezas individuales para después enviarlas al siguiente módulo una vez agrupadas en pallets). Se clasificaron a los módulos 3 y 5 como módulos tipo **WareHouse** y al módulo 4 como **Palletizer**.

4.1.3. Funcionamiento a Detalle de la funcionalidad de la aplicación (el Simulador)

En esta sección se explica a detalle la ejecución de la parte funcional del sistema, una vez que se ha generado el código de cada una de las clases que integran el simulador. De esta manera se ilustra el funcionamiento y la integración del código mediante el cual se modeló la celda de ensamble. La Figura 4. 6 ilustra la interacción del simulador con el resto de la aplicación, siendo configurado y ejecutado por el usuario desde la aplicación cliente, a través del servicio Web y la sesión que le corresponde, para mandar de regreso la aplicación cliente la lista de eventos generada en la simulación.

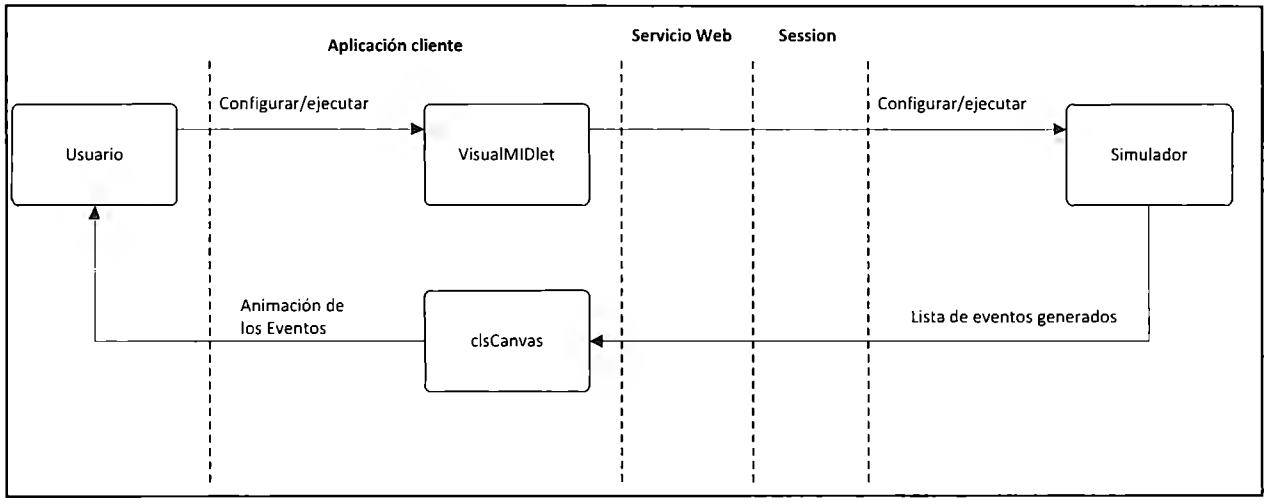


Figura 4. 6: Diagrama general del funcionamiento de la lógica funcional en la aplicación

La clase **Handler** deberá crear instancias, primero de la clase **Schedule** y después de los diferentes módulos implicados, ordenados del último en la línea de ensamble al primero (para poder cumplir con los constructores de cada módulo, que requieren una referencia al siguiente módulo como argumento de entrada), posteriormente se crea una instancia de la clase **SystemArrival**, dicha clase deberá instanciar la clase de tipo **Distribution** que utilizará, la clase tipo **Hanler** llamará a un método de esta clase que genera llegadas al sistema y a partir de éstas genera llegadas al primer módulo, lo que desencadenará en el funcionamiento de toda la línea de ensamble virtual. Posteriormente la clase tipo **Handler** llamará a una función de la clase **Schedule** que almacenará los diferentes estados del sistema de acuerdo a los objetos tipo evento

que le hayan enviado los diferentes módulos. Dicha información almacenada podrá ser utilizada para crear registros, obtener estadísticas y generar animaciones. La Figura 4. 7 representa un diagrama de secuencia simplificado de la lógica funcional.

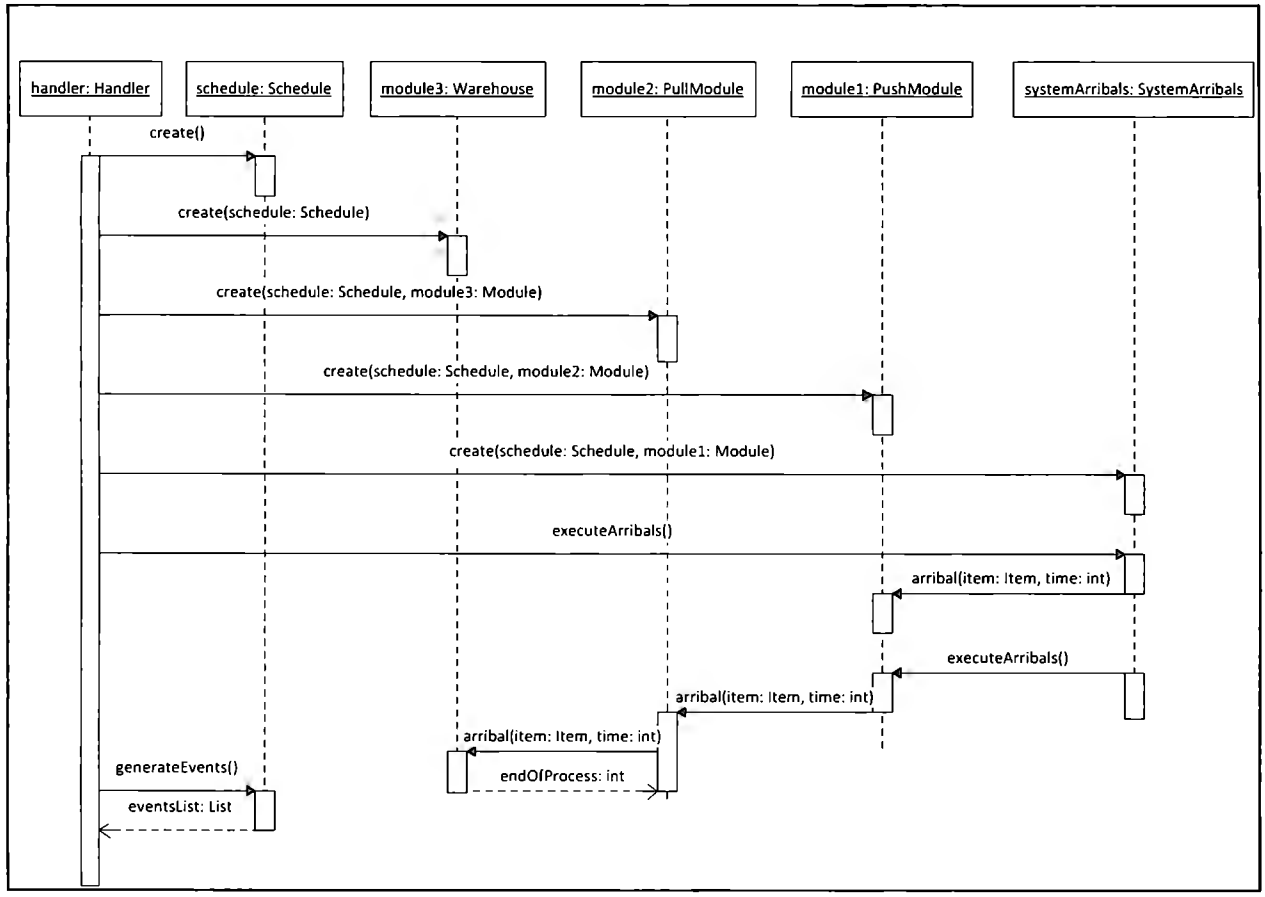


Figura 4. 7: Diagrama de secuencia de la lógica funcional

4.2. Implementación del Simulador Distribuido como Servicio Web

En esta sección se describe la implementación total del sistema distribuido, así como su funcionamiento para permitir a diferentes usuarios utilizar el simulador en equipos de trabajo de forma simultánea desde dispositivos móviles.

4.2.1. Descripción General de la Implementación del Lado del Servidor

El usuario va a ingresar a sesiones de grupo, cada sesión de grupo va a estar representada por una instancia de la clase **Session**, la cual va a llevar un registro de las actividades de los usuarios y va a servir de enlace entre estos y el simulador.

El servicio Web por su parte va a ser el enlace entre el usuario y la sesión, y va a llevar un registro de las sesiones existentes, de las cuales va a crear y administrar las instancias. A continuación se muestra el diagrama de secuencia de la aplicación web multiusuario vista desde el servidor. La Figura 4. 8 muestra el diagrama de secuencia simplificado de la aplicación completa vista desde el servidor.

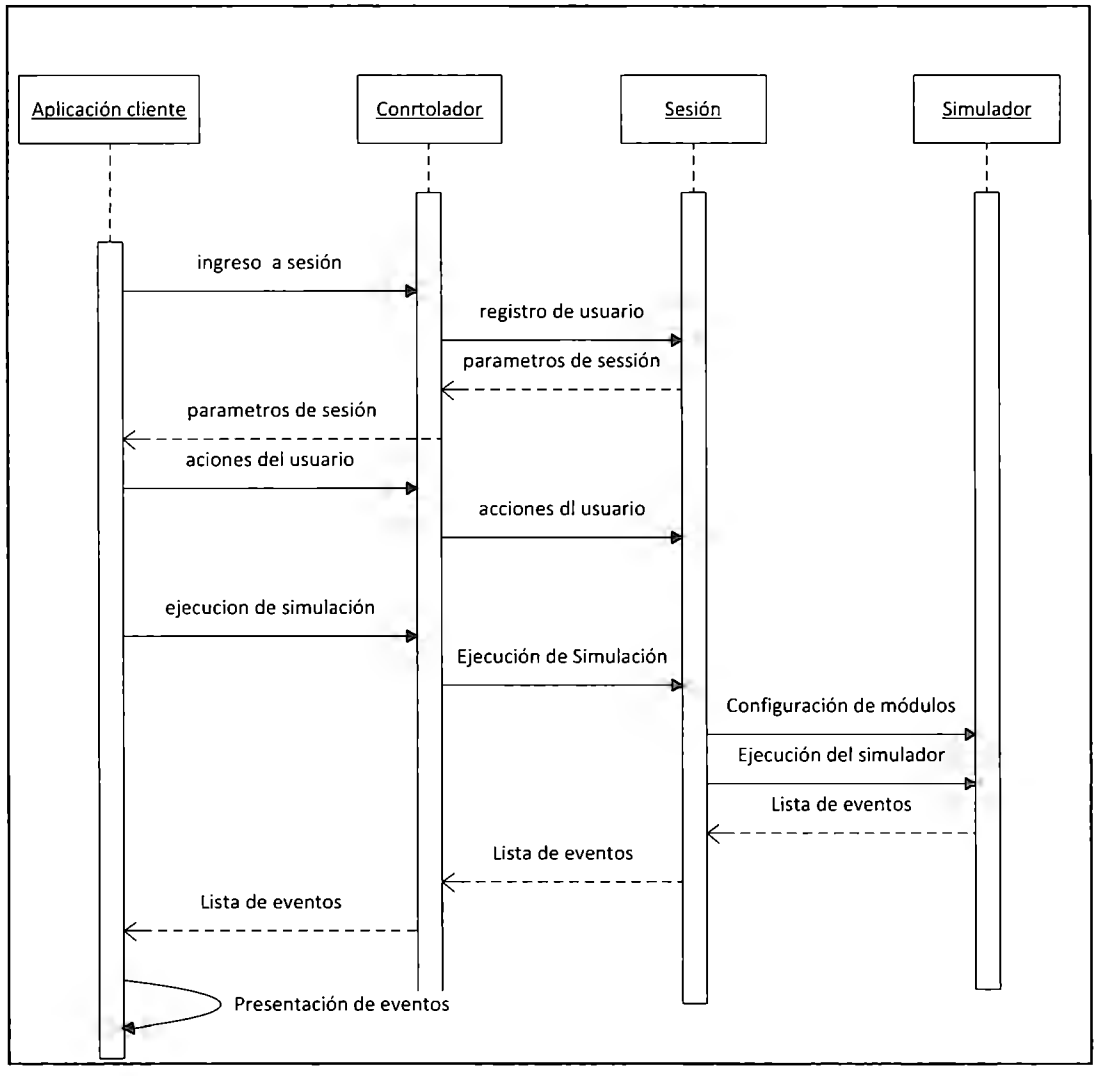


Figura 4. 8: Diagrama de secuencia de la aplicación multiusuario vista desde el servidor

4.2.2. Implementación del Paquete Session

El paquete sesión va a contener la clase sesión y todas las clases que ésta necesita para administrar las sesiones de grupos de usuarios. A continuación se enlistan las clases pertenecientes al paquete **Session** (la Figura 4. 9 va a mostrar el diagrama de clases parcial en torno a la clase Sesión):

- **Session**: el servicio Web va a crear una instancia de esta clase por cada grupo nuevo que sea creado por algún usuario. El servicio Web va a funcionar como un intermediario entre el usuario y el objeto sesión creado, de manera que muchos de las funciones publicadas por el servicio Web únicamente van a llamar un método de la clase sesión.
- **Module:** Se crea una instancia de ésta por cada módulo del simulador, esta clase va a contener un conjunto de parámetros de configuración y un indicador acerca de si está disponible o ha sido apartado por algún usuario. Dicha clase **Module** no es la misma clase Module del paquete **simulator** que es utilizada por el simulador, sino más bien una representación de los objetos de ésta última clase para coordinar su configuración entre los diferentes usuarios.
- **Status:** Se crea una instancia de esta clase por cada usuario, esta lleva un registro de si el usuario está listo y que módulo tiene actualmente apartado.
- **ConfigurationParser:** Va a encargarse de obtener la configuración del simulador a partir de la configuración que se le haya dado a los objetos tipo Module de la sesión correspondiente.

Cuando el usuario creé un nuevo grupo, el servicio Web va a crear una instancia de la clase **Session**, la cual va a almacenar en una lista y a asignarle los usuarios que estén registrados en dicho grupo por medio de un mapa usuario-sesión. El constructor de la clase **Session** se va a encargar de crear las cinco instancias de la clase Module, las cuales van a representar a los cinco módulos del simulador. La clase Module de este paquete va a contener todos los parámetros que pudiera tener cualquier objeto de la clase Modulo del paquete **simulator**, sin embargo únicamente pondrá a disposición del usuario aquellos parámetros que pertenezcan al módulo particular que dicho objeto tipo módulo represente. Igualmente, la clase **Session**

contará con una serie de métodos de configuración, los cuales servirán, cada uno, para configurar un módulo en particular (por ejemplo, se implementaron los métodos **configureModule4** y **configureModule5**, los cuales sirven para dar valores a los parámetros configurables del módulo 4 y el módulo5).

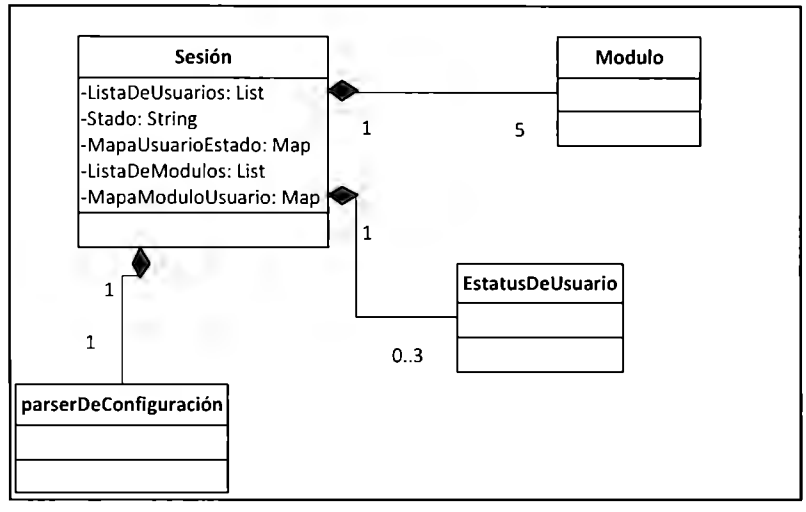


Figura 4. 9: Diagrama de clases parcial en torno a la clase Session

A la clase **Session** se le agregó también un mapa que relacionará al nombre del usuario con un objeto de tipo **Status**, dicho objeto va a contener una variable booleana que indique si el usuario se encuentra listo para que comience la ejecución de la simulación y una variable tipo **Module**, que es el módulo que tiene actualmente apartado (existe el módulo llamado *observer* el cual no va tener método de configuración, ni mostrará ninguna variable al usuario, este módulo es el único que puede ser seleccionado por más de un usuario y es el que tendrán seleccionado por *default todos los integrantes de la sesión*).

El método **doSimulation** será llamado cada vez que un cliente pida que se ejecute la simulación, antes de que se ejecute éste método se llamará al método **isGroupReady**, el cual iterará la lista de usuarios para revisar que cada uno de ellos se encuentre listo en su estatus,

de ser así, se procede a llamar al método **doSimulation**, de lo contrario no se realiza acción alguna.

La clase **Session** tiene una variable de tipo entero llamada **changesNum**, la cual va a representar el número de cambios que se han dado en la sesión, por ejemplo, si un usuario cambia su estatus de no listo (*notReady*) a listo (*ready*), entonces la variable **changesNum** aumentará en uno. De esta manera, la aplicación cliente se mantendrá revisando si cuenta con la última versión de la sesión al hacer uso (a través del servicio Web) del método **getChangeNum**, el cual va a regresar el valor de la variable **changesNum**, si dicha instancia de la aplicación cliente no cuenta con la última versión de la sesión va primero a revisar si ha comenzado la ejecución por medio del método **simulationStarted** para, de ser así, pedir la lista de eventos generada en la simulación, por medio del método **getResult** (ver Figura 4. 10).

En el caso de que no haya comenzado aún la ejecución de la simulación pedirá los status de sus compañeros enlistados en un **String**, por medio del método **getPartners** y llamará al método **getModules**, para pedir las configuraciones de los módulos en la sesión, también enlistadas en un **String**.

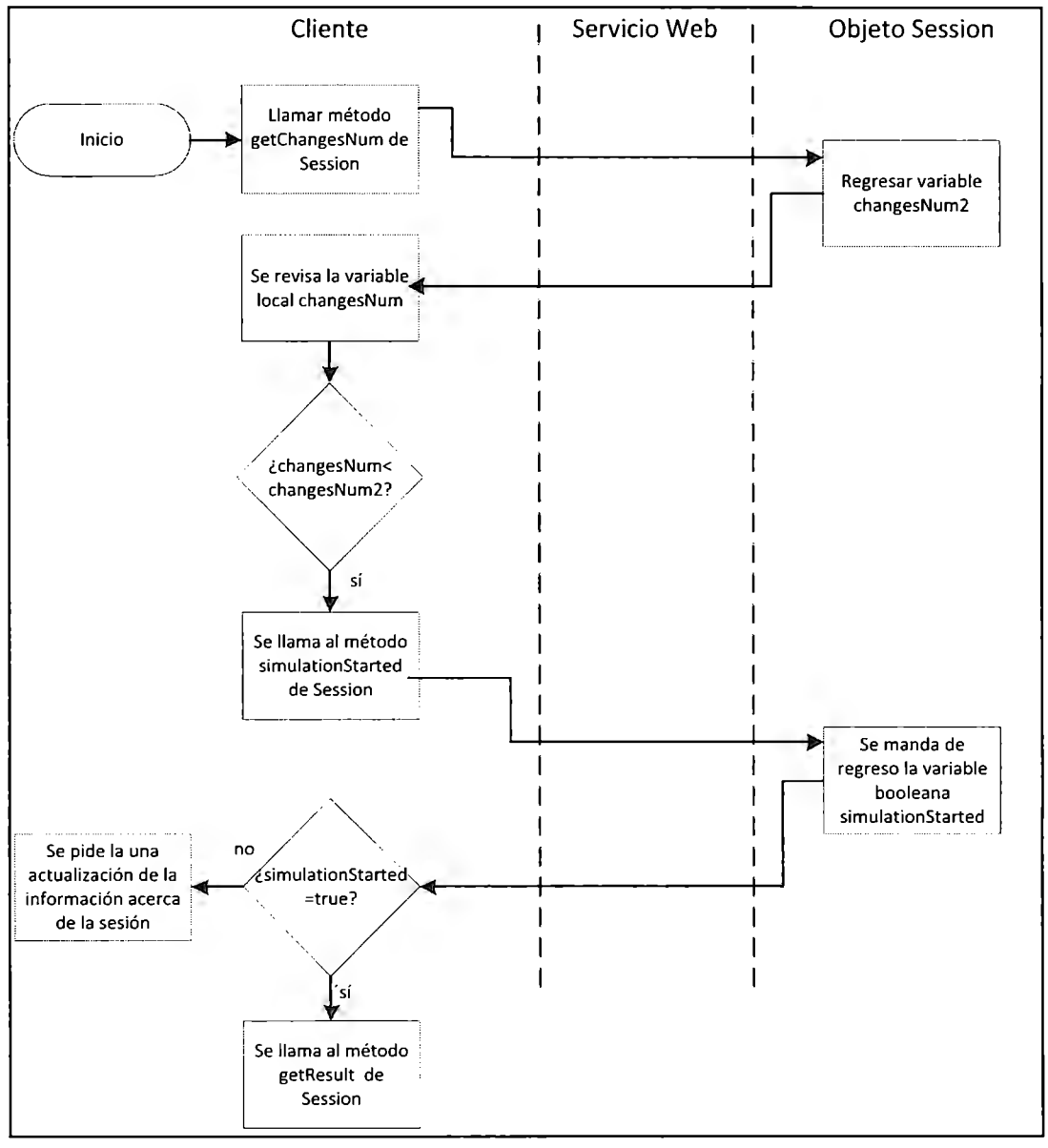


Figura 4. 10: Diagrama de flujo de la actualización que tiene el cliente de su información acerca del objeto Session

Otros métodos importantes de la clase **Session** se mencionan a continuación:

- **Método setStatus:** Mediante éste método el cliente puede cambiar su estatus de listo a no listo o viceversa.
- **Método takeModule:** Sirve para que el cliente seleccione alguno de los módulos que están disponibles, además de hacer disponible a los demás integrantes de la sesión el modulo que el usuario haya tenido seleccionado.

- **Métodos seter:** La clase **Session** cuenta con varios métodos de este tipo para permitir al cliente la configuración de los diferentes objetos tipo Module que contiene.
- **getPartners:** Devuelve al usuario una lista de **Strings** que contienen el nombre de cada usuario y su estatus en ese momento.
- **getModulesFeatures:** Manda al usuario una lista de los parámetros configurables del módulo que tiene seleccionado, para que pueda configurarlo.
- **getModules:** Manda al usuario la lista de objetos tipo **Module** pertenecientes a la sesión que aún estén disponibles, es decir, que no los tenga seleccionado ningún otro usuario.
- **ModulesDisplay:** Manda al usuario una lista con la configuración que actualmente tienen cada uno de los objetos **Module** pertenecientes a la sesión.

Cuando la clase **Session** manda a la aplicación cliente alguna lista, como la lista de módulos disponibles o la lista de usuarios presentes en la sesión con sus respectivos estatus, en realidad no manda un objeto de tipo lista sino un objeto de tipo **String**, cuyos elementos son separados por una diagonal, a partir del cual se van a crear las listas que dichos **Strings** representan.

4.3.3. Implementación del Paquete Simulator

El paquete **simulator** va a contar con todas las clases que componen el simulador, el objeto **Session** va a crear una instancia de su clase **Handler** para poder actualizar la configuración de los módulos en base a los parámetros determinados por los usuarios y ejecutar la clase principal del simulador una vez que los usuarios lo determinen.

4.3.4. Implementación del paquete Web Service

En esta sección se describen la programación del servicio Web y su implementación en el servidor.

4.3.4.1. Programación del Contenido del Paquete Web Service

El paquete Web Service va a contener al servicio Web. El servicio web sirve de enlace entre el cliente remoto y, la lógica del simulador y de administración de interfaces. Dicho servicio web expone las interfaces públicas de la lógica de sesión y lleva un registro de las diferentes sesiones existentes. Las interfaces del servicio web se describen a continuación:

- **getGroups ():** envía al cliente una lista de los diferentes grupos disponibles.
- **joinGroup (String user, String group):** Registra el nombre del cliente en un grupo.
- **createGroup (String groupName):** Crea una sesión para el Nuevo grupo con el nombre dado por el usuario.
- **getChangesNum (String group):** La aplicación cliente del usuario manda una petición del número actual de cambios que se han realizado en su sesión, de manera que si este es mayor que el número de cambios que tiene registrados, pedirá nuevamente las listas de usuarios, status de los usuarios y módulos (con su configuración correspondiente).
- **getMembers (String group):** Manda al cliente una lista con los miembros que se encuentran en el grupo en el que está registrado el usuario.
- **setReady (boolean isReady, String user, String group):** Cambia el status del usuario, de listo (*ready*) a no listo o viceversa.
- **takeModule (String module, String user, String group):** Asigna el modulo determinado por el usuario a su nombre de usuario.
- **getModules (String group):** Devuelve al usuario una lista con los módulos disponibles.
- **displayModules (String group):** Devuelve al usuario una lista con la configuración de cada módulo perteneciente a su grupo o sesión.
- **setModule1Yellow (String user, String group, String fa, String tba, String ma, String dist):** Configura los parámetros del módulo 1 relacionados con la llegada de piezas amarillas.
- **setModule1Blue (String user, String group, String f, String tba, String dist):** Configura los parametros del módulo 1 relacionados con la llegada de piezas azules.

- **setModule4 (String user, String group, String process1, String process2):** Configura los parámetros del modulo4 en relación con el tiempo requerido para agrupar el primer piso del pallet, y con el tiempo requerido para agrupar el segundo.
- **setModule5 (String user, String group, String process1, String process2):** Configura los parametros del módulo 5 relativos a las velocidades horizontal y vertical del vehículo que almacena los pallets en el almacén final.
- **signOut (String sesión, String user):** Retira al usuario del grupo al que pertenece, ya sea porque decidió salirse de la sesión o porque ya fue ejecutada la simulación en la sesión a la que pertenece.
- **resetStatus (String session):** Una vez ejecutada la simulación se llama a este método para que establezca la configuración inicial para los parámetros que sea necesario.
- **isGroupsFull ():** regresa al usuario una variable booleana indicándole si hay espacio para más grupos, de manera que no se generen demasiados en el servidor.
- **isGroupReady(String group):** revisa que todos los miembros del grupo estén listos antes de ejecutar la simulación y enviar a todos ellos la lista de eventos generados en dicha simulación.

El servicio web almacena las sesiones existentes en una lista de sesiones, además de brindar al usuario los métodos **createGroup**, para crear sesiones nuevas (el código 4.5 muestra el pseudocódigo de dicho método), y **joinGroup** (el código 4.6 muestra el pseudocódigo de dicho método), para unirse a sesiones ya existentes. La mayoría de los demás métodos solamente llaman a un método correspondiente de la sesión a la que pertenece el usuario.

Código 4.5: Pseudocódigo del método createGroup

```
@WebMethod(operationName="createGroup")
Public String createGroup(String user, String groupName){
//se revisa primero si se alcanzado el número máximo de grupos existentes
Session session=new Session(groupName);
```

```
        sessions.add(session);
        joinGroup(user, groupName);
        //crea y registra la nueva sesión, y registra al usuario en esta
    }
```

Código 4.5: Pseudcódigo del método joinGroup

```
    @WebMethod(operationName="jpinGroup")
    Public boolean joinGroup(String user, String group){
            Session=SessionSesName.get(group);
            boolean accepted=Session.register(user);
    //si el grupo está lleno o está ejecutando una simulación será rechazado
    //Se registra al usuario en la sesión determinada
    }
}
```

4.3.4.2. Implementación del Servicio Web en el Servidor

El servicio Web se creó por medio de la herramienta automática del ambiente de desarrollo NetBeans para la creación de servicios Web. Dicha herramienta aporta los lazos (*ties*) y el motor RPC, que se van a encargar de manejar todos los detalles de las invocaciones remotas al servicio Web, además de generar un archivo de Lenguaje de Descripción de Servicio Web (WSDL por sus siglas en inglés), que va a describir los métodos que el servicio Web hace accesibles y que va a ser usado por el desarrollador de la aplicación cliente para generar un cliente del servicio web.

Se tuvo que implementar el servicio Web, junto con el resto del software del lado del servidor, en un servidor con dirección pública, ya que los dispositivos móviles comúnmente no se conectan a internet desde una red local, cómo es el caso de los dispositivos BlackBerry Pearl 8130 con los que se contaba para realizar las pruebas, y para que pueda ser accedida la aplicación desde cualquier sitio.

Se usó un contenedor Tomcat, creado por la Fundación de Software Apache (ASF por sus siglas en inglés) para publicar el servicio Web desde el equipo con dirección pública, y se le asignó un puerto de dicho equipo. De este modo el desarrollador de la interfaz de usuario, para crear una aplicación cliente de dicho servicio, solo tiene acceder a la descripción del servicio web en el lenguaje de descripción de servicios Web disponible en la dirección donde se implementó el servicio Web.

4.4. Implementación de la Aplicación Cliente

En esta sección se describe la programación de la aplicación cliente y su implementación en dispositivos BlackBerry.

4.4.1. Programación de la Aplicación Cliente

Tanto la interfaz como la presentación de los resultados fueron realizadas usando exclusivamente APIs pertenecientes a MIDP 2, la suite del MIDLet se compone de las siguientes clases:

- **VisualMIDLet:** Es la clase principal (que extiende a la clase **MIDlet**), la cual se encarga iniciar y finalizar la aplicación completa. Además, se encarga también de desplegar las diferentes presentaciones (*displays*) que el usuario va a ver, así como los diferentes botones en pantalla mediante los cuales dicho usuario va a poder interactuar con la aplicación.
- **ItemListener:** Implementa la clase **ItemStateListener**, Esta clase es utilizada para registrarla en una forma o presentación de manera que sea notificada de cualquier evento relacionado con cualquier elemento (*item*) perteneciente a la forma en cuestión. De esta manera, el usuario puede cambiar su status (a listo o no listo) con solo apretar un botón e, igualmente, seleccionar y apartar un módulo de línea de ensamble con solo seleccionarlo de una lista sin tener que utilizar comandos demás.
- **AuxiliarThread:** Esta clase implementa la clase **Runnable**, de manera que se le puede asignar un thread (hilo de ejecución) únicamente para la ejecución de su método **run**. Esta clase se encarga de revisar los cambios que se han realizado en la sesión del

usuario, y, de existir cambios, actualiza la información que el cliente tiene acerca de ella. También se encarga de descargar el resultado de la simulación, y de crear y ejecutar el **GameCanvas**, que es la clase encargada de desplegar la animación de la simulación.

- **clsCanvas:** Esta clase extiende a la clase **GameCanvas** (perteneciente al paquete para desarrollo de videojuegos incluido en la especificación MIDP) y se encarga de desplegar paso a paso los eventos resultantes de la simulación, esto ya sea en tiempo real o a una velocidad mayor de acuerdo a la elección del usuario.

La Figura 4. 11 muestra el diagrama de secuencia de la aplicación completa vista desde el cliente. Ahí se puede ver como la clase VisualMIDlet inicia la ejecución del método principal de la clase ThraedAuxiliar, y como éste se va a mantener actualizando la información de la clase VicualMidlet, además de iniciar la generación de la animación por parte de la clase clsCanvas una vez que obtenga la lista de eventos. La clase clsCanvas finalizará la ejecución del método principal de la clase ThreadAuxiliar. También se puede observar que la clase VisualMidlet manda directamente al controlador (situado en el servidor) las peticiones hechas por el usuario.

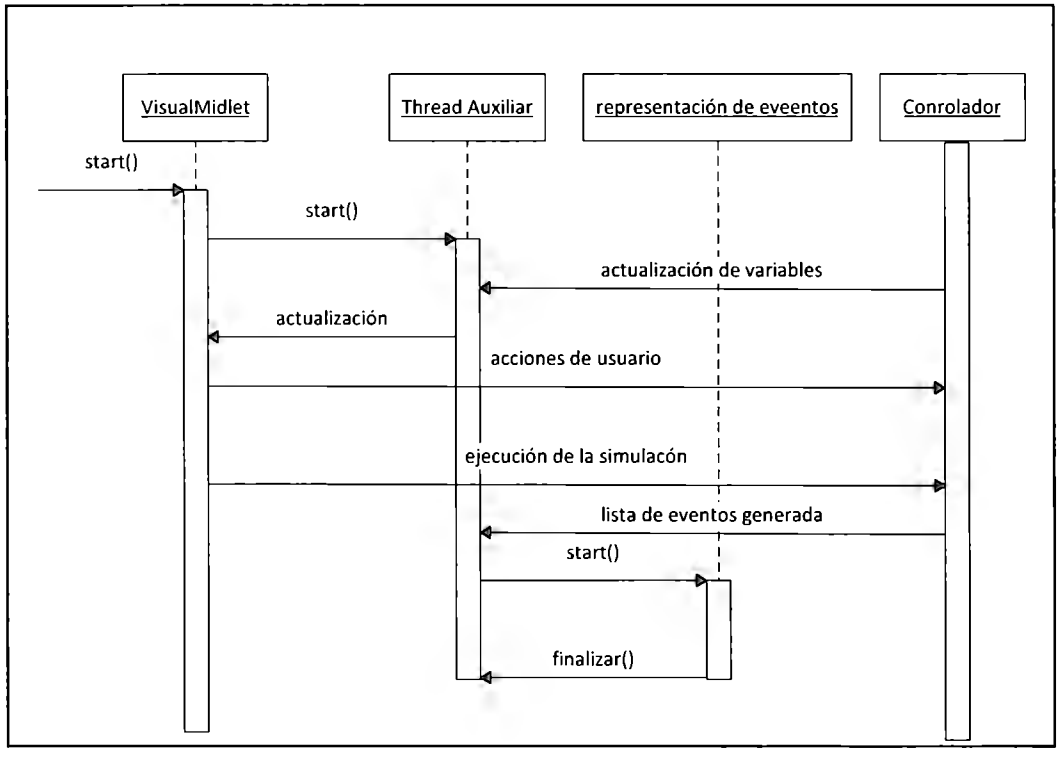


Figura 4. 11: Diagrama de secuencia de la aplicación multiusuario vista desde el cliente

Para la programación de la aplicación cliente primero se utilizó el ambiente gráfico para el desarrollo de aplicaciones móviles de NetBeans, el cual genera bloques de código automáticamente a partir de un diagrama, para crear las diferentes pantallas que se mostrarán al usuario y la secuencia que estas tendrán, así como para agregar botones para el menú de comandos. La Figura 4. 12 muestra el diagrama hecho en NetBeans para el desarrollo de la aplicación cliente.

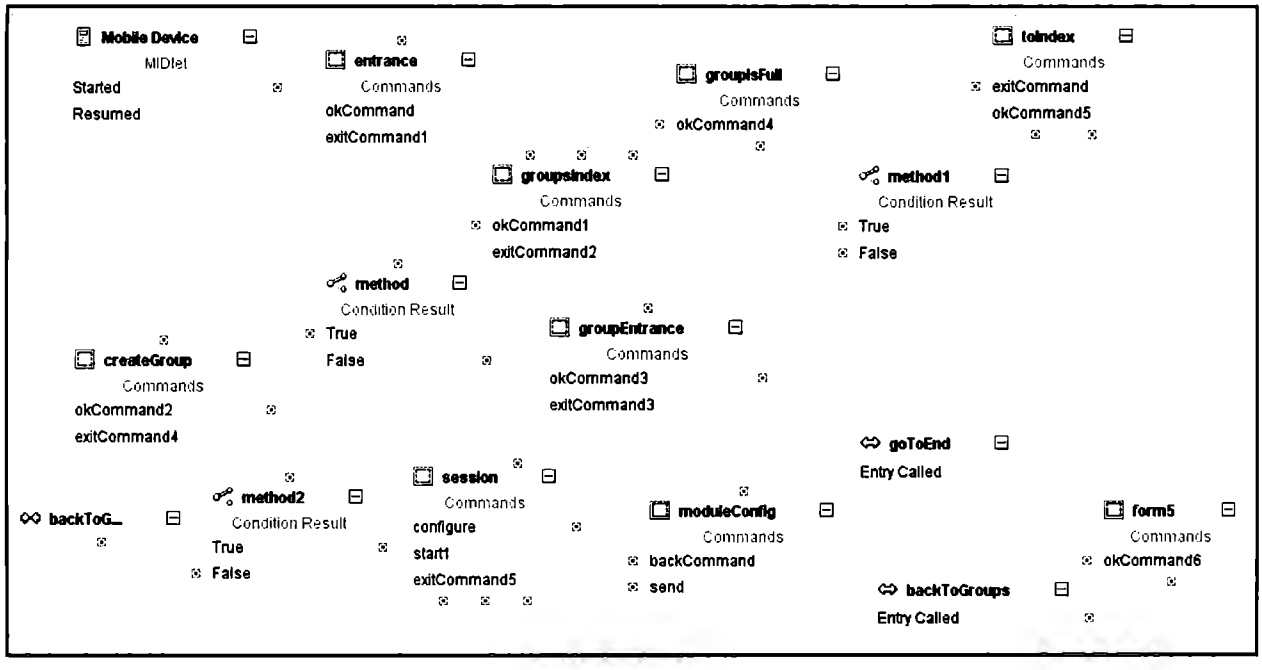


Figura 4. 12: Diagrama en NetBeans de la aplicación cliente

A continuación de describe la función de cada una de las pantallas:

- **entrance:** Es la primera pantalla que el usuario ve en la aplicación, este le pide el nombre de usuario que va a usar.

Figura 4. 13: Pantalla "entrance"

- **groupIndex:** Presenta al usuario una índice de los grupos actualmente existentes, en esta pantalla el usuario va a poder elegir a que grupo ingresar o elegir crear un grupo nuevo. Si el usuario eligió un grupo es llevado a la pantalla groupEntrance y si eligió crear un grupo nuevo es llevado a la pantalla createGroup.

Figura 4. 14: Pantalla "groupIndex

- **createGroup:** En esta pantalla se le pide al usuario que ingrese el nombre del grupo que desea crear, sí existen demasiados grupos es llevado a la pantalla form5, en la cual se le informará que existen ya demasiados grupos y se procede a guiarlo de nuevo al índice de grupos. Por otro lado, si no existen demasiados grupos, se creará el grupo del usuario y éste es llevado a la pantalla **session**.

Figura 4. 15: Pantalla "createGroup"

- **session:** Una vez que el usuario haya ingresado a un grupo o creado uno nuevo es enviado a esta pantalla, ésta le muestra los usuarios que están presentes en la sesión, la configuración que tengan en ese momento los módulos del simulador, así como elementos mediante los cuales podrá determinar su estatus, pedir que sea ejecutada la simulación y elegir un módulo para configurarlo. Una vez terminada la simulación los usuarios que se encuentren presentes en esta serán llevados de vuelta al índice de grupos.

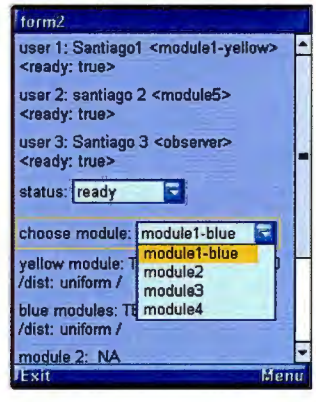


Figura 4. 16: Pantalla "session"

- **moduleConfig:** Sirve para que el usuario pueda configurar el módulo que tiene seleccionado.

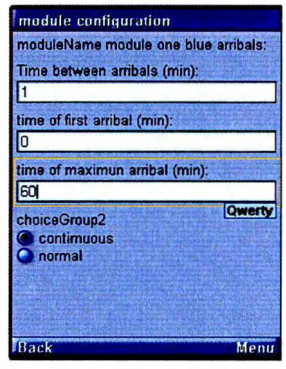


Figura 4. 17: Pantalla "moduleConfig"

El código generado automáticamente por Net Beans a partir del diagrama mostrado en la Figura 4. 12 tuvo que ser ampliamente modificado para permitirle interactuar con los objetos remotos tipo sesión y, a su vez, con el simulador, también situado en el simulador.

El objeto tipo **AuxiliarThread** va a ser instanciado e inicializado por la clase **VisualMIDlet**. Dicho objeto va a tener su propio hilo de ejecución y se va a mantener en un *loop* revisando si ha habido cambios en la sesión para, de ser así, bajar la información de ésta para mantenerse actualizado, mediante dicha información va a modificar las variables que el usuario ve en la pantalla **session**. Sus métodos más importantes de describen a continuación:

- **run:** Este método va a contener las instrucciones que serán ejecutadas en el hilo de ejecución asignado al objeto tipo **AuxiliarThred**. Dicho método se mantendrá en un *loop* revisando si ha habido cambios, para de ser así pedir las actualizaciones al objeto **Session** del lado del servidor, por medio del servicio Web (ver código 4.5).
- **actualize:** Por medio de este método el objeto **AuxiliarThread** va a bajar la información actual del objeto **Session** y a modificar la pantalla **Session** de acuerdo a dicha información (ver código 4.6). Antes de bajar dicha información revisa si ya se ha iniciado la simulación, para de ser así terminar el *loop* para que el objeto **AuxiliarThread** salga del *loop* y pida al servicio Web que obtenga, del objeto **Session**, la lista de eventos generados. Una vez que tiene la lista de eventos generados crea un

objeto **clsCanvas** y le pasa dicha lista y el control de la pantalla del dispositivo móvil, para que genere una animación de la simulación (ver código 4.7).

Código 4.5. Pseudocódigo parcial de la instrucción while del método **run** del **AuxiliarThread**

```
While(isOn){
   Try{ //Se deben manejar las posible excepciones que se den al invocar métodos remotos
      changesNum2=simServ.getChangesNum(group); //pide el número de cambios que ha habido
   }catch(Exception e){}
   if(changesNum2>changesNum){//¿ha habido más cambios de los que se tienen registrados?
       Actualice();
       /*Si ha habido un cambio en la sesión desde la última vez que se actualizó la información acerca
       de ésta se volverá a realizar la actualización.*/
   }
 }
/*Si la simulación ha comenzado se le da un valor de falso a la variable isOn y se termina de ejecutar la
 instrucción while*/
```

Código 4.6: Pseudocódigo de método **actualize** del **AuxiliarThread**

```
Actualice(){
    Boolean simulationStarted=false;
    /* Se actualiza toda la información de la sesión acerca de los estatus de los compañeros y las
    configuraciones de los diferentes modulos*/
   try{
      simulationStarted=simServ.simulationStated(group);
   }catch(Exception e){}
  If(simulationStarted){
      /* se verifica si se está en la etapa de ejecución de la simulación
       IsOn=false
```

```
        /*esto  hace que finalize el loop en el cual se actualiza la información de la sesión, para que proceda,
        de ser el caso, a ejecutar la simulación, y a bajar los eventos generados en  la simulación.
    }
    Else{
        //se actualiza el display (la pantalla) de la sesión
    }
}
```

Código 4.7: Pseudocódigo del método run del **AuxiliarThread** posterior a la instrucción while

```
If(isStarter){
    //si el usuario inició la ejecución del simulador va a ser el único en generar los resultados
    Try{
        simServ.doSimulation();
    }catch(Exception e){}
}

Try{
    String resultString=simServ.getResultString(group);
}catch(Exception e){}
/*se genera una lista de eventos a partir del String que se recibe como resultado  de la simulación
  Y se procede a generar la animación de la simulación*/
```

La clase **clsCanvas** se va a encargar de desplegar una animación en pantalla de los eventos generados en la simulación en base a un reloj de simulación, así como de manejar las peticiones que el usuario haga en relación al cambio en la velocidad a la que se desenvuelve la animación. Sus métodos principales son los siguientes:

- **run:** Se va a mantener en un *loop* haciendo avanzar el reloj de simulación y revisando qué eventos se han dado en el tiempo que ha avanzado dicho reloj, para tomarlos de uno

en uno en orden cronológico para desplegar imágenes en pantalla que los representen. En dicho *loop* revisará también las teclas que ha presionado el usuario para determinar la velocidad a la que el reloj de simulación irá avanzando.

- **getKeyStates:** Mediante este método va a poder detectar las teclas del dispositivo móvil que ha presionado el usuario.

El código 4.8 muestra un pseudocódigo parcial de la clase **clsCanvas**.

Código 4.8: Pseudocódigo de la clase **clsCanvas**

```
while(isRunning){
    //Se mantiene funcionando hasta que el cliente lo determine
    while(isRunning&&(localTime<=lastEventTime)){
        /*mientras haya eventos por representar y el cliente no haya interrumpido la animación
        Se ejecutará este loop*/
        If(index<sizeOfEventsList){
            Event partEvent=eventsList[index];
            String partTimeS=partEvent.time;
            partTime=Integer.parseInt(partTimeS);
        }
        If((partTime<=(localTime+100))&&(index<sizeOfEventsList)){
            /* Si el siguiente evento que se tome de la lista de eventos, determinado por la variable tipo
            entero "index", se encuentra dentro de un rango de 100 milisegundos a partir del tiempo
            actual en milisegundos en el reloj de simulación, determinado por la variable localTime, se
            actualizará la representación gráfica del sistema en base a dicho evento*/
            index++;
        }
        else{
            localTime+=100;
            Thread.sleep(100*animationRate);
```

```
            /*Si el siguiente evento de la lista de eventos tiene un tiempo 1000 milisegundos mayor al
            tiempo actual se aumenta el tiempo actual 100 milisegundos y no se realizan cambios*/
        }
        If((iKey&RIGHT_PRESSED)!=0){
            //se acelera la velocidad de la animación si se aprieta RIGHT en el teclado
            }
        If((iKey&LEFT_PRESSED)!=0){
            //se desacelera la velocidad de la animación si se aprieta LEFT en el teclado
            }
        If(iKey&FIRE_PRESSED){
            //se abandona la animación si se aprieta FIRE en el teclado
            }
    }
}
```

4.4.2. Implementación de la Aplicación Cliente como Cliente del Servicio Web

Para convertir la aplicación móvil en cliente del servicio Web de la aplicación se utilizaron las herramientas que el ambiente de desarrollo NetBeans ofrece para hacerlo automáticamente. Dichas herramientas generaron automáticamente un *stub* o *proxy*, que va a representar el servicio Web remoto y a través del cual se van a invocar métodos en dicho servicio Web, de manera remota. El *stub* generado va a ser utilizado para convertir la invocación de métodos al stub en invocaciones remotas al servicio Web, usando el Protocolo Simple de Acceso a Objetos (SOAP por sus siglas en inglés).

4.4.3. Implementación de la Aplicación Móvil en Dispositivos BlackBerry

Para la implementación de ésta aplicación primero se utilizó el simulador estándar de dispositivos móviles incluido en el ambiente de desarrollo NetBeans y luego se tuvieron que realizar modificaciones con un simulador de BlackBerry para adaptarla a la pantalla de dichos dispositivos, especialmente en la parte de la representación gráfica, ya que las distancias entre las

imágenes es más cercana en dichos dispositivos, de manera que la representación creada con el simulador de Net Beans no servía directamente para los dispositivos BlackBerry.

El código Java en BlackBerry debe ser almacenado en archivos .COD, que es un formato propietario, para ello se utilizó la herramienta rapc perteneciente al ambiente de desarrollo en Java para BlackBerry desde la línea de comando, esto a partir de los archivos JAD y JAR incluidos en la suit del Midlet de la aplicación cliente. A continuación se muestra el comando utilizado:

Rapc import="C:BlackBerryJDE3.6.libnet_rim_api.jar" codename=VisualDlet –midlet jad=VisualMIDlet.jad VisualMIDlet.jar

Una vez que la aplicación se encuentra almacenada en un archivo .COD está disponible para ser bajada y utilizada desde los dispositivos BlackBerry.

Capítulo 5

Resultados

El implementar el simulador en el servidor permitió que el desarrollo de la aplicación fuera independiente de las limitaciones encontradas en el desarrollo de aplicaciones para dispositivos móviles, en cuanto a capacidades de memoria y procesamiento, así como lenguajes de programación disponibles. Aunque aún las redes inalámbricas son considerablemente lentas, si se implementa una arquitectura de tipo cliente-servidor, donde la aplicación cliente se encargue de crear y administrar la interfaz de usuario, y la presentación de los resultados, se puede disminuir la cantidad de datos que se requiera enviar y recibir, y de ese modo reducir el efecto que pueda tener un flujo lento de datos entre ambos dispositivos. En el caso de la aplicación presentada, las actualizaciones en el dispositivo cliente acerca de la información en el servidor únicamente se daban cuando se había realizado un cambio en dicha información, con lo cual se reducía el flujo de información considerablemente.

A pesar de que el simulador funciona de forma adecuada, en algunas ocasiones, debido a la falta de experiencia, se tuvo que generar código redundante, así como código que no tenía que ser integrado de usar adecuadamente poliformismo, esto debido a que al desarrollar un simulador de este tipo la primera vez resulta complicado determinar todos los aspectos que es necesario tomar en cuenta, siendo estos últimos evidentes hasta el momento de la programación.

Po otro lado, la portabilidad de la aplicación cliente resultó limitada, ya que, el aspecto de las pantallas en la interfaz de usuario, que fue creado por medio de las herramientas automatizadas de NetBeans para generar pantallas, usando únicamente APIs pertenecientes a MIDP y el simulador de dispositivos móviles de NetBeans, lució considerablemente distinto cuando se ejecutó dicha aplicación en los dispositivos BlackBerry. Esto además de las modificaciones que tuvieron que ser realizadas en la parte del código encargada de generar las animaciones, al desplegarse las imágenes más cerca entre ellas en los dispositivos Blackberry. Del mismo modo

se esperaría que sucediera lo mismo al implementar la aplicación cliente para tipos adicionales de dispositivos móviles, e inclusive, tal vez, para otros modelos de BlackBerry.

Detalles de la Implementación

El software del lado del servidor se instaló en un servidor con procesador Intel Xeon de 2Ghz, 4GB de memoria RAM, 279 GB de disco duro y sistema operativo Windows 2008 Server, haciendo uso del JDK 1.6. El servicio Web se publicó en un servidor Apache Tomcat 6.0.20, haciendo uso del protocolo SOAP.

Los dispositivos móviles que se utilizaron para La aplicación cliente fueron BlackBerry Pearl 8130, con sistema operativo v4.5.0.173 (plataforma 3.4.0.55), configuración de CLDL 1.1, perfil MIDP 2.0 y JTWI versión 1.0, que utilizaron el servicio 3G de IUSACELL para acceder a Internet, y a través de ahí acceder al servicio Web. Ver Figura 5. 1.

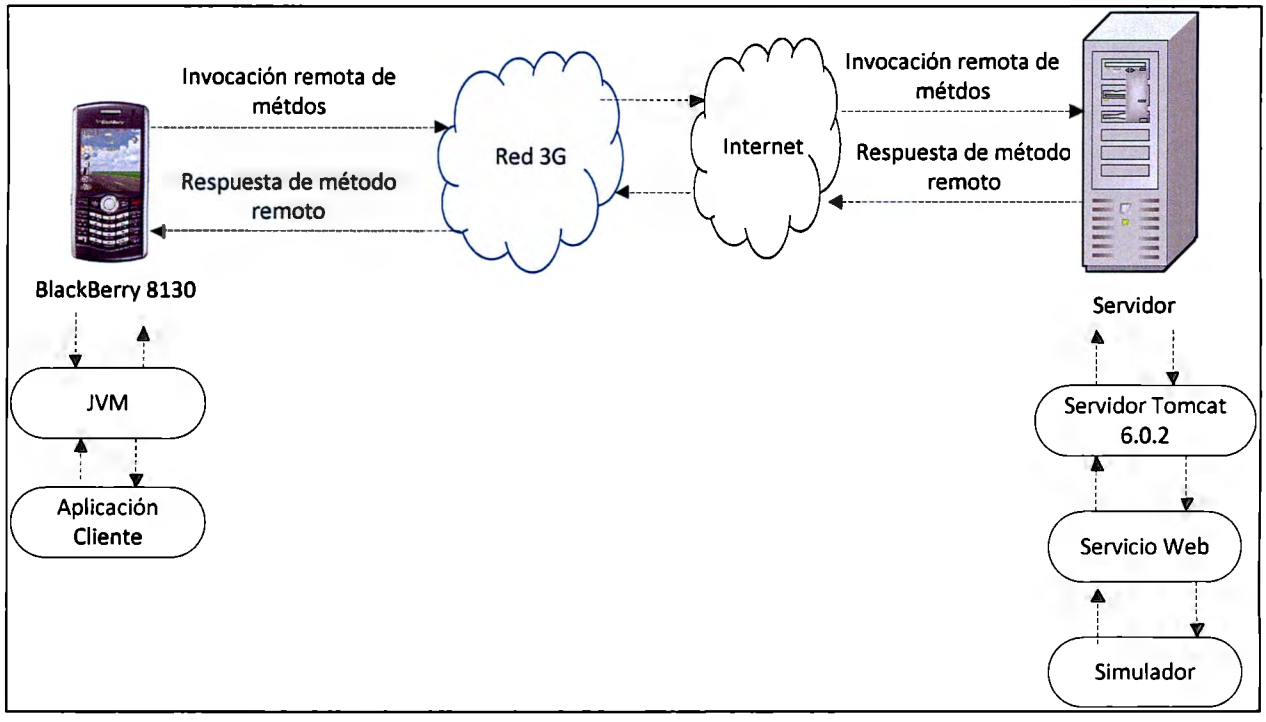


Figura 5. 1: Detalles de la implementación

Diseño de Pruebas del Simulador

Se realizaron pruebas de caja negra de tal manera que se analizaran los diferentes escenarios que pudieran generar variaciones en el funcionamiento de la línea, de modo que se pudiera determinar que el simulador sigue las mismas reglas que la celda de ensamble en su funcionamiento. Pressman define a las pruebas de caja negra de la siguiente manera:

"Las pruebas de caja negra son las que se aplican a la interfaz del software. Una prueba de este tipo examina algún aspecto funcional de un sistema que tiene poca relación con la estructura lógica interna del software"[26]

Se plantearon los escenarios de tal manera que se pudiera observar que el funcionamiento del simulador de la celda de ensamble fuera adecuado, independientemente del orden en que llegaban las piezas de color azul y amarillo, de la cantidad de piezas que llegaban al sistema, del tiempo que transcurría entre llegadas o si las llegadas al sistema seguían a una distribución normal. Adicionalmente se realizaron pruebas del funcionamiento de la línea en sus límites de operación, por ejemplo, con el almacén final siendo llenado (esta parte revisó en parte mediante el registro de texto generado por el simulador) o con el módulo 3 siendo llenado, modificando la velocidad con la que llegaban las piezas a los almacenes de éste (de este modo se corroboró el funcionamiento del almacén de repuesto de dicho módulo).

Las pruebas serían satisfactorias al mantenerse funcionando el simulador siguiendo fielmente las reglas que sigue la celda de ensamble. Se hizo énfasis en revisar que el simulador siguiera las reglas bajo las cuales operaba la celda de ensamble, ya que al funcionar de misma manera que la celda de ensamble la precisión prácticamente dependería de los tiempos tomados del sistema

[26] Pressman, RS 2005, *Ingeniería del Software: Un enfoque práctico*, sexta edición, traducido del inglés por Murrieta Murrieta, JE, Pineda Rojas, E & Campos Olguín, V, McGraw-Hill, México

real. En cuanto a los tiempos, se revisó que cada módulo tuviera tiempos de trabajo acordes con la celda de ensamble.

Se realizaron una cantidad considerable de pruebas de la aplicación (alrededor de 100), de las cuales, las más representativas se describen a continuación.

Pruebas con un Solo Usuario

- Tiempo entre llegadas constante, con únicamente llegadas de piezas azules, con tiempo entre llegadas de 1, 2 y 3 minutos, con tiempo máximo de llegada (llegada de la última pieza) de 60 minutos. La Figura 5. 2 muestra una simulación con llegadas de piezas amarillas cada minuto.
- Tiempo entre llegadas constante con únicamente llegadas de piezas amarillas, con tiempo entre llegadas de 1, 2 y 3 minutos, con tiempo máximo de llegada (llegada de la última pieza) de 60 minutos.
- Tiempo entre llegadas constante tanto para piezas amarillas como para piezas azules, con el tiempo de llegada de la primera pieza azul mayor al tiempo de llegada de la última pieza amarilla
- Llegadas de piezas amarillas y azules alternadas, con tiempo entre llegadas constante para ambos casos.
- Tiempo de llegada en base a una distribución normal con solo piezas amarillas.
- Tiempo de llegada en base a una distribución normal tanto con piezas azules como con piezas amarillas.

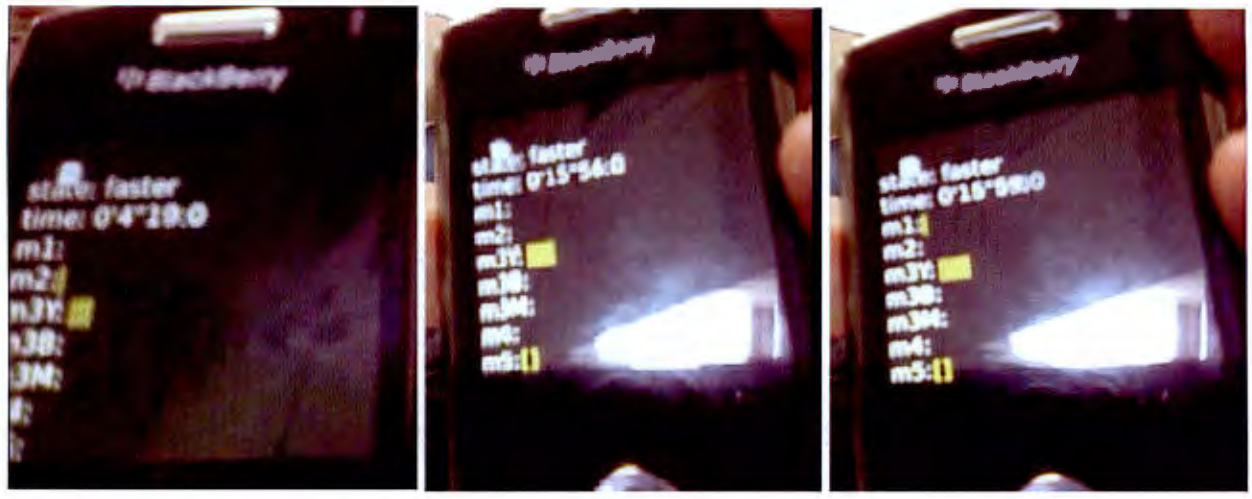


Figura 5. 2: simulación de un sólo un usuario, con llegadas de piezas amarillas cada minuto

Pruebas con tres Usuarios

- Con el tiempo de llegada de la primera pieza azul mayor al tiempo de llegada de la última pieza amarilla, con tiempo entre llegadas constante para ambos tipos de pieza.
- Tiempo entre llegadas constante sólo para piezas amarillas.
- Tiempo entre llegadas constante únicamente para piezas azules.
- Llegada de piezas azules alternadas con llegadas de piezas amarillas, con tiempo entre llegadas constante para ambos tipos de piezas.
- Llegadas de piezas azules y amarillas en los mismos tiempos, con tiempo entre llegadas constante para ambos tipos de piezas. La figura 5.3 muestra una simulación hecha por tres usuarios con llegadas amarillas y azules intercaladas, con los mismos tiempos de llegada para ambos colores.
- Tiempo entre llegadas con distribución normal, tanto para piezas amarillas como para piezas azules.

Figura 5. 3: Simulación hecha por tres usuarios, con llegadas azules y amarillas intercaladas

Los escenarios anteriores sirvieron para corroborar que los eventos generados en la simulación son coherentes para diversos ambientes representativos, y que lo mismo puede decirse independientemente de la cantidad de usuarios que se encuentren participando en una sesión en específico. La secuencia y los tiempos de los eventos generados fueron satisfactorios.

Se debe mencionar que, además de las pruebas realizadas al simulador completo, durante el desarrollo se hicieron pruebas tanto de las funciones de cada módulo (de caja negra) como de caja blanca al interior del código, de cada uno de los módulos a integrar a éste, para procurar agregar dichos módulos ya con un funcionamiento adecuado, de este modo facilitando las pruebas e integración del simulador completo.

Proceso de Validación de la lógica funcional

La lógica de negocios fue corroborada al momento del desarrollo de cada uno de sus módulos, ya que el objetivo en ese entonces fue generar código que representara adecuadamente el modelo que se creó de cada uno de los módulos de la línea de ensamble real, incluidos los tiempos

tomados de cada uno de los procesos que tienen lugar en dichos módulos. Sin embargo, resultó importante revisar el funcionamiento en conjunto de dichos módulos al ser integrados en la celda de ensamble virtual. A continuación se describe la lógica de validación.

Es importante tomar en cuenta que, al ir aumentando la cantidad de llegadas de piezas que se simulen, al realizarse la modelación del sistema a detalle, lo cual principalmente le da robustez al simulador, las discrepancias en tiempos de décimas de segundos se irían convirtiendo en segundos a lo largo de la simulación. También se debe mencionar que el proceso de validación es susceptible a tener cierto nivel de error, ya que, al probar la celda en corridas de alrededor de 40 minutos, comúnmente se llegan a presentar errores humanos en la medición y errores en el funcionamiento de la celda de ensamble, por ejemplo, llega a suceder que piezas queden momentáneamente atascadas, o que sean colocadas en lugares o almacenes equivocados (e.g. piezas azules colocadas en el almacén para piezas amarillas), e inclusive pudieran darse pequeñas variaciones en algunos tiempos dentro de la misma celda de ensamble. A continuación se enlistan condiciones a observar en el proceso de validación:

- Condición 1. La secuencia de trabajos en la celda de ensamble debe ser adecuada.
- Condición 2. Se deben mantener las reglas de operación del módulo 3.
- Condición 3. Se deben mantener las reglas de operación del módulo 4.
- Condición 4. Se deben seguir las reglas de operación del módulo 5.
- Condición 5. Los tiempos deben ser similares a los de la celda de ensamble.

Condición 1. La secuencia de trabajos en la celda de ensamble debe ser adecuada: Cada pieza debe seguir el orden de operaciones de la celda de ensamble (cada pieza debe pasar por cada uno de los módulos en el orden correcto).

Condición 2. Se deben mantener las reglas de operación del módulo 3: Cada vez que uno de los almacenes del módulo 3 tenga 10 pieza almacenadas, deberá sacar esa misma cantidad de piezas de una en una. Se sacan primero las 10 piezas de un almacén antes de comenzar a sacar

piezas del otro. El hecho de que un almacén se encuentre sacando piezas no debe implicar que deje de recibir piezas.

Condición 3. Se deben mantener las reglas de operación del módulo 4: Se deben sacar pallets compuestos por las 10 piezas anteriormente recibidas, las 10 piezas deben ser del mismo color, excepto cuando se llene el almacén de repuesto del módulo 3.

Condición 4. Se deben seguir las reglas de operación del módulo 5: Se debe mantener el orden de almacenamiento. Los primeros pallets amarillos se deben colocar en el primer piso del sub-almacén 1 (repisa 1), en orden del lugar más próximo al más lejano con respecto al módulo anterior, las azules en el segundo piso de la repisa 1, también del lugar más próximo al más lejano. Una vez que se hayan llenado las hileras mencionadas de la repisa 1, se debe proceder a llenar cada piso del almacén 2, del lugar más próximo al más lejano, en orden ascendente e independientemente del color de las piezas de los *pallets*. El tercer piso de la repisa 1 no es utilizado, por lo menos cuando no se llega a utilizar el almacén de repuesto del módulo 3, sin embargo, provisionalmente se consideró que fuera llenado, después de que haya sido llenada la repisa 2, por *pallets* de cualquier color. El tiempo en que los *pallets* sean colocados en una de las repisas debe depender de la cercanía del lugar donde sean colocadas al punto de llegada al módulo.

Condición 5. Los tiempos deben ser similares a los de la celda de ensamble: Se deben considerar los tiempos de los módulos susceptibles a generar las variaciones más considerables en los tiempos de operación de la celda de ensamble (el módulo 3 y el módulo 2) o de la celda completa, el resto de los módulos funciona con tiempos constantes o poco variables, sobre todo para las posibles condiciones que puedan ser determinadas por el usuario y verificables en la celda de ensamble. La variación en los tiempos debe estar a lo mucho alrededor del 10% de error, debiendo considerarse la cantidad de llegadas simuladas, ya que a medida que se simule mayor cantidad de piezas mayor será el efecto de los posibles errores de medición, ya sea

nuevos, cometidos al momento de hacer la validación, u originales, cometidos al momento de modelar el sistema.

De este modo se considera como aceptable el resultado de una simulación si cumple con las diferentes condiciones mencionadas anteriormente. La Figura 5. 4 muestra un diagrama de flujo del proceso de validación.

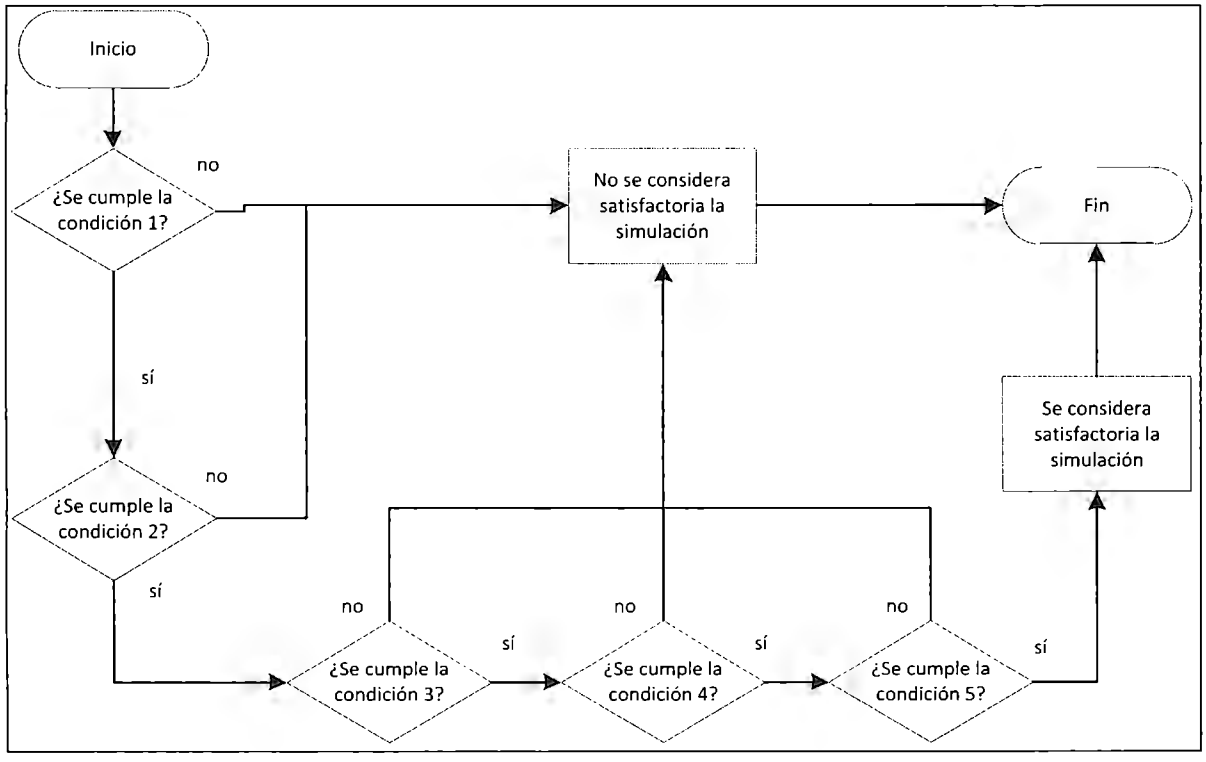


Figura 5. 4: Diagrama de flujo del proceso de validación

Para la validación se observaron los resultados de la simulación por medio de un archivo de texto, donde es impresa la ocupación a detalle de cada uno de los módulos del simulador después de cada evento y la ubicación detallada de las diferentes piezas y pallets, también por medio de la impresión en pantalla de la lista de eventos generada en la simulación. La Figura 5. 5 muestra una porción del archivo de texto utilizado en la validación de la lógica funcional.

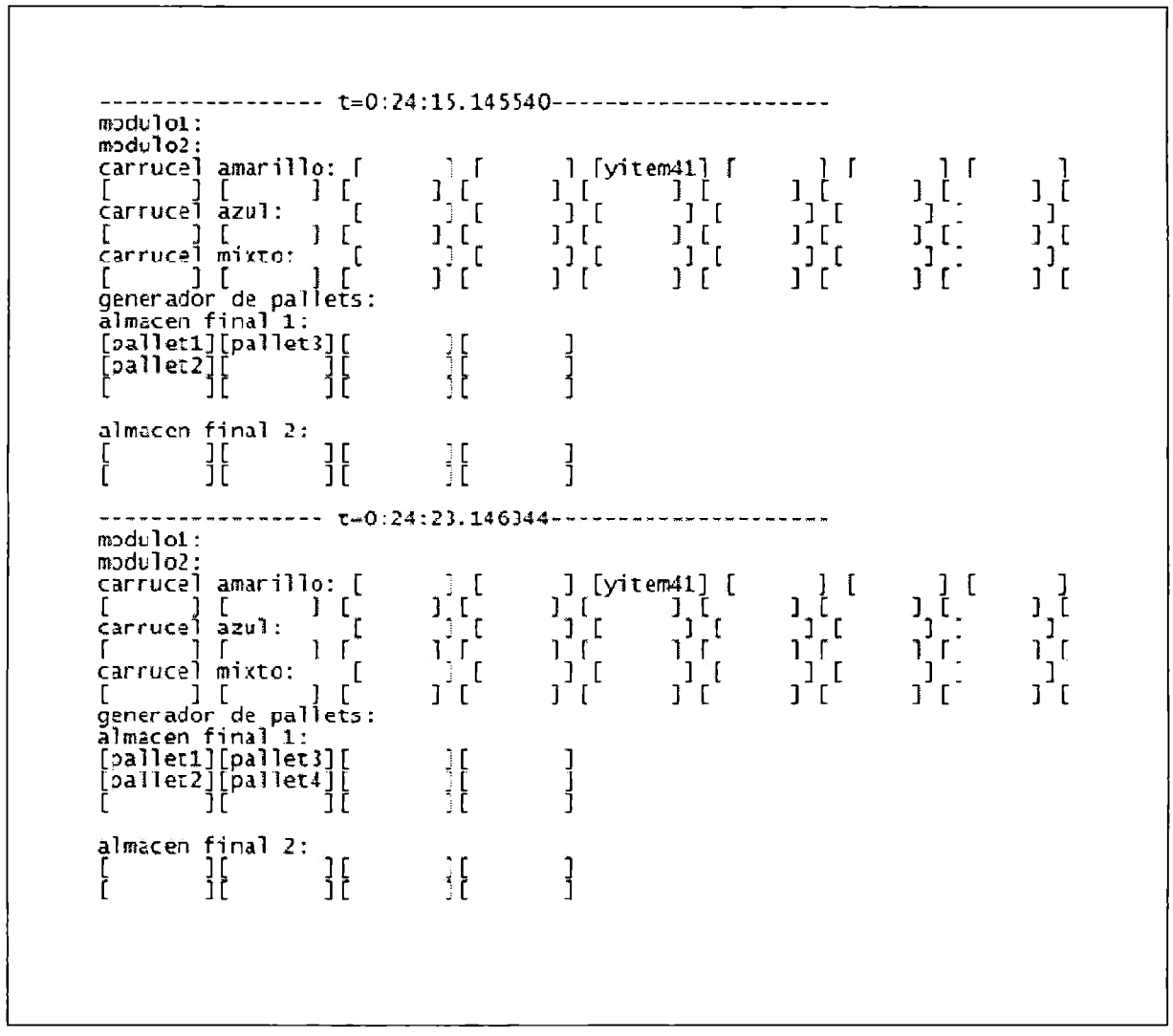


Figura 5. 5: Imagen del archivo de texto donde se imprime la ocupación de los módulos

Ejemplos de Validación de la Lógica Funcional

Ejemplo 1: Validación con tiempos de llegada iguales para piezas azules y amarillas, con un tiempo entre llegadas constante de un minuto:

Se revisó el tiempo promedio entre la primera y la última pieza en salir del módulo 3, de los grupos de diez piezas sacados por dicho módulo. Se obtuvieron los siguientes resultados:

- Tiempo promedio registrado en la línea de ensamble: 1 minuto con 14 segundos.

- Tiempo promedio registrado en el simulador: 1 minuto con 8 segundos.
- La diferencia entre los tiempos registrados: 6 segundos.
- Porcentaje de error: 9.6%

Se revisó el tiempo promedio de la diferencia entre la salida de los pallets del módulo 4 y su colocación en el módulo 5, se obtuvieron los siguientes resultados:

- Tiempo promedio registrado en la celda de ensamble: 8.4 segundos.
- Tiempo promedio registrado en el simulador: 8.0 segundos.
- Diferencia entre los tiempos: 0.4 segundos.
- Porcentaje de error: 4.6%

La diferencia entre los tiempos registrados en la línea de ensamble fueron menores a los 10 segundos y representaron un porcentaje de tiempo menor a 10%, por lo que se puede considerar satisfecha la condición 5, descrita anteriormente en esta misma sección.

Las demás condiciones de aceptación fueron observadas directamente en la celda de ensamble, siendo completamente satisfactorias, de modo que se determinó que la simulación fue satisfactoria, la Tabla 5. 1 presenta el desempeño de la simulación percibido.

	Condición 1	Condición 2	Condición 3	Condición 4	Condición 5
La simulación es satisfactoria	sí	sí	sí	sí	sí

Tabla 5. 1: Desempeño de la simulación del ejemplo 1 de acuerdo a los criterios de aceptación definidos

Ejemplo 2: Validación con llegadas únicamente de piezas amarillas, de dos en dos cada minuto de manera constante durante 40 minutos:

Se revisó el tiempo promedio de la diferencia entre la salida de la primera pieza en salir del módiulo3, de los grupos de diez piezas sacados por dicho módulo, y la colocación del *pallet* correspondiente en el almacén final (módulo 5). Se obtuvieron los siguientes resultados:

- Tiempo promedio registrado en la celda de ensamble: 2 minutos con 16 segundos.
- Tiempo promedio registrado en la simulación: 2 minutos con 10 segundos.
- Diferencia entre tiempos promedio: 6 segundos.
- Porcentaje de error: 5%.

Se revisó el tiempo promedio de la diferencia entre la salida de los *pallets* del módulo 4 y su llegada al módulo 5, se obtuvieron los siguientes resultados:

- Tiempo promedio registrado en la celda de ensamble: 7.6 segundos.
- Tiempo promedio registrado en el simulador: 6.2 segundos.
- Diferencia entre los tiempos: 1.4 segundos.
- Porcentaje de error: 18%

En este caso el porcentaje de error se encuentra alejado del 10%, sin embargo, debe considerarse que se simuló el ingreso de 80 piezas y que en este caso los tiempos son muy pequeños, de modo que, aunque se registró un porcentaje de error de 18%, el error solamente era de 1.4 segundos. De modo que 18% de error se puede considerar aceptable.

Las demás condiciones de aceptación fueron observadas directamente en la celda de ensamble siendo completamente satisfactorias, de modo que se determinó que la simulación fue satisfactoria, la **¡Error! No se encuentra el origen de la referencia.** presenta el desempeño percibido de la simulación.

	Condición 1	Condición 2	Condición 3	Condición 4	Condición 5
La simulación es satisfactoria	sí	sí	sí	sí	sí

Tabla 5. 2: Desempeño de la simulación del ejemplo 2 de acuerdo a las condiciones de aceptación definidas

Desempeño del simulador en modo multiusuario

En el caso de las pruebas realizadas la velocidad de la red inalámbrica no tuvo ningún efecto negativo en el desempeño de la aplicación ya que realmente se percibía el intercambio de información y el despliegue de la animación como prácticamente inmediatos. Además, el desempeño de las redes inalámbricas es algo que se espera que mejore drásticamente en los próximos años, aunque en las pruebas realizadas no provocó deterioro alguno de la aplicación, funcionando en algunos casos mejor que en redes locales inalámbricas, por medio de simuladores de dispositivos móviles ejecutados en computadoras portátiles y de escritorio.

Capítulo 6

Conclusiones y Trabajo a Futuro

La implementación del modelo utilizado permite una transición relativamente rápida de aplicaciones creadas para PC a aplicaciones creadas para dispositivos móviles, al permitirle al desarrollador enfocarse únicamente en la creación de una nueva interfaz de usuario, una nueva presentación de los resultados obtenidos y de acoplar dichos componentes de software nuevos a la aplicación original, en este caso en un servidor. Sin embargo esto último no es necesario si se implementa una arquitectura basada en servicios Web.

El aplicar una arquitectura basada en servicios Web permite crear interfaces de usuario que sean independientes de la lógica funcional, de manera que, en el caso del simulador, si se quiere crear una nueva interfaz de usuario, o, una nueva versión para PC o para iPhone (que no cuenta con la versión de Java para Mobile), el desarrollador de dichas interfaces no necesitará familiarizarse con la lógica funcional y únicamente deberá conocer las interfaces publicadas por el servicio Web para crear un cliente que las utilice, sirviendo como vínculo entre la lógica funcional y la interfaz de usuario. Del mismo modo, esta arquitectura agilizaría el desarrollo de otros posibles componentes a agregar a la aplicación, como pudieran ser agentes tutores inteligentes cuya interacción puede ser implementada por medio de servicios Web.

La versión modificada del paradigma basado en eventos para la creación de la lógica funcional, si bien permitió una visualización y un análisis de pruebas más simple del modelo implementado, no resultó en un método muy eficiente en cuanto al uso de memoria y la cantidad de código generado.

A pesar de que se podía observar el comportamiento de la simulación módulo por módulo de manera separada, sin intercalar demasiados métodos de diferentes módulos en la ejecución del simulador, al hacer uso de listas de eventos, en la mayoría de los módulos no se creó un código conciso y en ocasiones se produjo algo de redundancia al ser similares algunas listas de eventos de salida de algunos módulos a las listas de eventos de llegada del siguiente módulo. Esto además de que todos los eventos eran registrados en una lista de eventos global (la lista de eventos a enviar al cliente).

Por otro lado, dada la naturaleza de esta versión modificada del paradigma orientado a eventos, el uso de este esquema en algunos otros casos, concretamente en los casos en los que se deba modelar un sistema en el que el flujo de información o de entidades pueda ser cíclico, resultaría complicado si se quiere mantener el mismo grado de modularidad de la aplicación. Al ejecutarse la lógica de cada módulo por separado y en orden cronológico, para introducir, por ejemplo, la posibilidad de que una entidad regrese del módulo dos al uno, requeriría de introducir información al modelo que fuera compartida por ambos módulos mencionados, de esta manera, los dos módulos quedarían fuertemente acoplados, volviéndose considerablemente más complicado el implementar la lógica funcional de forma distribuida o en una arquitectura basada en componentes, de ser necesario.

El código generado permite la generación de código nuevo a partir de las clases abstractas creadas y del código encargado del control de la aplicación, así como la reutilización de las clases no abstractas en caso de que se quiera modelar un sistema similar en el que, por ejemplo, existan más módulos de tipo "*push*", otro módulo de tipo "*palletizer*" o inclusive llegadas de pernos al sistema en base a alguna distribución estadística, utilizando la clase **SystemArrivals**.

En cuanto a la implementación, se pudo observar que, al haber diferencias en el uso de las clases y el despliegue en pantalla con respecto al simulador de NetBeans basado en MIDP, que las aplicaciones entre dispositivos que soportan dicho estándar no son directamente portables, si

bien los cambios que se requiere efectuar no son drásticos sí es de esperarse que se requieran corregir errores al utilizar la misma aplicación en diferentes tipos de dispositivos.

La aplicación desarrollada puede ser utilizada para que los alumnos de Ingeniería Industrial del Tecnológico de Monterrey puedan complementar el uso de la línea de ensamble, a la cual tienen acceso limitado, al poder realizar pruebas adicionales del sistema, de manera que, al usar la línea de ensamble real puedan optimizar su uso al haber realizado anteriormente pruebas con el simulador. El simulador permite además reducir la escala de tiempo permitiendo al alumno probar diversos escenarios sobre periodos largos de tiempo de manera práctica, sin tener que invertir demasiado tiempo en ello. El simulador también permite generar llegadas al sistema en base distribuciones estadísticas.

Además, la línea de ensamble actualmente no permite que se realicen cambios fácilmente en su configuración, el simulador, en cambio, permite que los alumnos analicen varios escenarios en el funcionamiento de la línea de ensamble, haciendo posible que el usuario cambie la velocidad con la que se realizan varias tareas en la línea de ensamble, al reducir su tiempo de duración. Fácilmente, de ser necesario, se podría permitir al usuario cambiar otros parámetros, como pudieran ser la velocidad de las bandas o, inclusive, el orden en que los *pallets* son colocados en el almacén final.

El permitir que varios usuarios puedan utilizar el simulador en grupos, desde dispositivos móviles, permite que se puedan crear prácticas colaborativas en las que el alumno pueda participar en prácticamente cualquier lugar, en el momento en que lo decida. De este modo, la interacción con sus compañeros de trabajo y la liberad de trabajar con el simulador, prácticamente, dónde y cuándo el alumno lo deseé, pueden mejorar considerablemente la experiencia de aprendizaje. Esto además de la experiencia y familiaridad que irían adquiriendo los alumnos con la hoy nueva tecnología móvil.

6.1. Limitaciones

Los dispositivos móviles con que se contaba para probar el sistema carecían de acceso a redes locales, por lo que se conectaban directamente al servicio de internet IUSACELL, por lo cual necesitaban tener acceso a la aplicación desde internet, por lo que se montó la aplicación en un servidor del departamento de Ingeniería y Arquitectura del Tecnológico de Monterrey. El acceso al simulador era complicado por lo que la realización de las pruebas en los dispositivos BlackBerry fue más lento e incierto que lo esperado, siendo que inclusive se llegó a buscar y probar alternativas. En algunas ocasiones la celda de ensamble del laboratorio también quedaba fuera de servicio.

Otro aspecto que dificultó las pruebas fue la decisión de escoger el intervalo de llegadas al sistema usando minutos como unidad, debido a eso el realizar las pruebas mismas pruebas en la celda de ensamble real que con la aplicación completa terminada resultara ser muy complicado.

La madurez del proceso de desarrollo de simuladores, más aun simuladores de este tipo repercutió en la calidad del código generado, lo cual complicó las pruebas, el mantenimiento y las modificaciones. También la documentación y el orden en el desarrollo en general sufrieron algunas repercusiones por lo mismo.

La modelación de la celda de manufactura fue hecha a un detalle suficientemente amplio como para seguir funcionando independientemente de cualquier cambio que se dé eventualmente en el funcionamiento de celda de ensamble real, cambiando parámetros determinados del simulador o haciendo modificaciones menores para integrar reglas nuevas de funcionamiento, como pudiera ser el orden en que se almacenan los *pallets* en el módulo 5. El elaborar el simulador con tal nivel de detalle permitiría fácilmente integrar una representación gráfica detallada de la simulación. Sin embargo, actualmente no es posible generar cambios en la configuración de la celda den ensamble y la animación que se hizo para la versión de este proyecto representa la línea de ensamble a un nivel de detalle considerablemente menor en comparación con la información que se genera en las simulaciones.

6.2. Proposición de Trabajo a Futuro

En esta sección se mencionan diversos trabajos a futuro propuestos, en base al proyecto descrito en el presente trabajo de tesis.

El simulador genera información suficiente para crear gráficas e inferir parámetros estadísticos de interés acerca del funcionamiento de la línea de ensamble, como pudieran ser los tamaños de cola promedio o el tiempo promedio de las piezas en el sistema. El tamaño de la pantalla de los dispositivos móviles fue una limitante en cuanto al contenido que se puso a disposición del usuario en el desarrollo de este proyecto, del mismo modo, la comunicación entre usuarios tuvo que ser limitada y se tuvo que eliminar una pizarra de mensajes que originalmente estaba en la interfaz de usuario. De modo que el diseño de una interfaz de usuario práctica y funcional que brinde al usuario información complementaria a la observación directa de la simulación, y que además encaje adecuadamente en la pantalla de los dispositivos móviles es una importante opción de trabajo a futuro.

Sería adecuado que la aplicación contara con pantallas que el usuario pudiera seleccionar para obtener información acerca de los módulos, de la línea de ensamble en general o de temas relacionados, funcionando la aplicación como un pequeño ambiente de aprendizaje, dándole al simulador un aspecto y una funcionalidad más académicos.

Agregar a la aplicación un módulo (implementado como servicio Web) que contenga un tutor inteligente, de manera que, ya sea el usuario o el servicio Web, mediante el cual este accede al simulador, le mande los eventos generados en la simulación, para que los analice y explique al usuario porqué la configuración que dio a la máquina no es adecuada y, mediante un modelo del conocimiento del usuario pueda recomendarle material de apoyo a éste.

Agregar, del mismo modo, al simulador un módulo de análisis que le permita desplegar al usuario un conjunto de gráficas e información en relación a los diferentes parámetros con los que se puede medir el desempeño de la línea de ensamble.

Agregar al simulador un módulo que implemente algoritmos para determinar programas de producción óptimos en base a parámetros de producción determinados por el usuario.

Ampliar y mejorar la presentación de los eventos resultantes de la simulación, de manera que el usuario pueda seleccionar entre diferentes vistas del simulador a desplegar en pantalla, de modo que, por ejemplo, el usuario, al seleccionar el módulo uno, vea una animación exclusivamente de dicho módulo operando, de manera que tenga una mejor comprensión de lo que sucede en la simulación.

Crear un simulador más abstracto, que utilizara una interfaz gráfica (que puede ser creada usando el API de videojuegos de J2ME) para permitirle al usuario crear sus propios simuladores. Esto permitiría al usuario contar con una herramienta de modelación simple en cualquier lugar y en cualquier momento (*on the go*) e inclusive poder compartir modelos de simulación con compañeros de trabajo a distancia. De manera que un alumno pueda probar simulaciones de sus propios modelos en el autobús de la escuela o inclusive un empleado pueda eventualmente mostrar una nueva configuración de una planta de producción a su jefe mientras este toma sus vacaciones. Se requeriría gran cantidad de trabajo y *expertice* para generar una aplicación con todas las herramientas que ofrece ofrecen aplicaciones similares, como es el caso de Arena, sin embargo, dicha aplicación sería utilizable desde dispositivos móviles y por varios usuarios trabajando en equipo, esto además de que evitaría la necesidad de adquirir licencias.

Conectar el simulador al sistema de control de la línea de ensamble de manera que el usuario pueda utilizar el simulador como interfaz para la configuración de parámetros. La celda de

manufactura configura los PLCs utilizados por la línea de ensamble, se podría buscar la forma de importar la configuración del simulador a dicho programa para proporcionar a los usuarios un medio práctico de modificar el sistema para probar diferentes escenarios.

Apéndice A: Manual de Usuario

A. Cómo Instalar la Aplicación en el Dispositivo Móvil

I. Descargar la aplicación A Través del Aire (OTA por sus siglas en ingles).

1) Ingrese al navegador Web del dispositivo.
2) Ingrese a la dirección donde se encuentra disponible la aplicación.
3) Descargar aplicación.

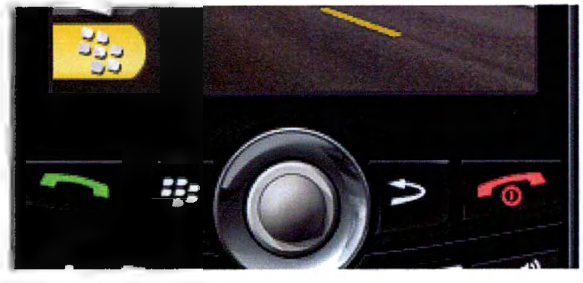
II. Descargar la aplicación desde la computadora.

1) Descargar el ambiente de desarrollo de Java (JDE por sus siglas en inglés) de RIM.
2) Conectar el dispositivo BlackBerry a la computadora mediante un cable mini-USB.
3) Ir a la línea de comandos.
4) Abrir el directorio donde se encuentra el archivo VisualMIDlet.cod de la aplicación.

5) Ejecutar el comando: javaloader –usb load VisualMidlet.cod

B. Cómo Ingresar a una Sesión

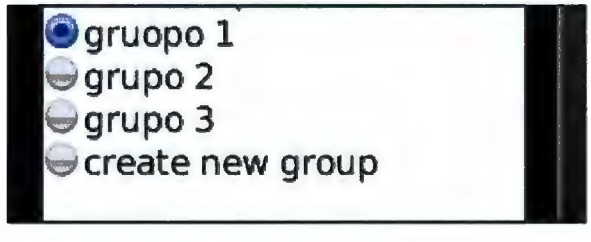
1) Seleccione el menú de Aplicaciones en la pantalla del dispositivo BlackBerry.



2) Seleccione la aplicación con el nombre de "VisualMidlet" y oprima la tecla de *fire*.

3) Al presionar *fire,* se le pedirá su nombre de usuario. Introdúzcalo y oprima "OK" en el menú de comandos. Al hacer esto será llevado al menú de grupos, que contiene todos los grupos existentes donde se podrá registrar.

4) Si desea ingresar a un grupo selecciónelo y oprima *fire* (será llevado a la sesión seleccionada). Si desea crear uno nuevo seleccione la opción de "*create new group*" situada al final del menú de grupos.

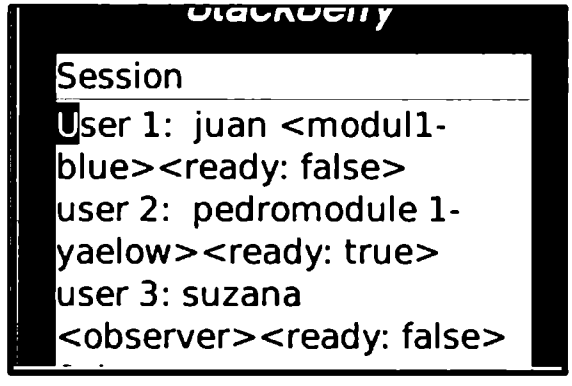


5) Si usted seleccionó la opción de *"create new group"* se le pedirá que ingrese el nombre del grupo nuevo, ingréselo y oprima *fire* (será llevado a la nueva sesión creada).

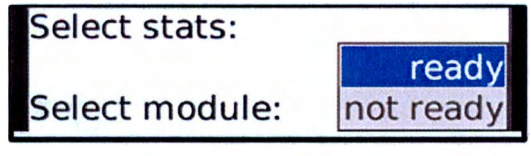
C. Sesión de grupo

En la sesión de grupo usted verá los siguientes componentes:

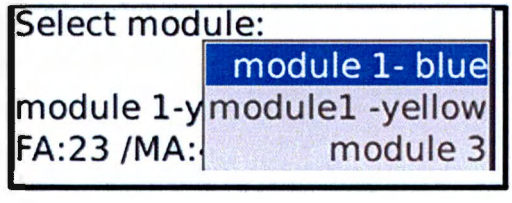
- Una lista con los usuarios que se encuentran actualmente dentro de la sesión, cada nombre de usuario será acompañado por una breve descripción acerca de su estatus y qué módulo tiene seleccionado



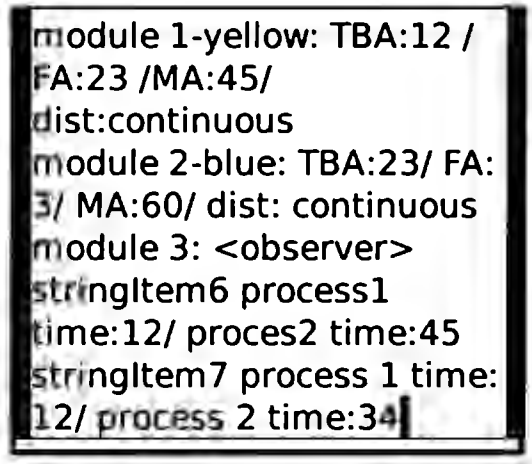
- Un menú donde podrá elegir su estatus, para determinar si se encuentra listo para que se ejecute la simulación.



- Un menú donde podrá seleccionar un módulo, de manera que pueda configurarlo. Únicamente aparecerán los módulos que no estén seleccionados por otro usuario.



- Una lista con los diferentes módulos del simulador y su configuración actual.



Restricciones:

- Cada módulo únicamente va a poder ser seleccionado por un solo usuario en todo momento.

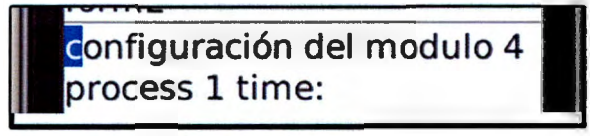
C.1. Configuración de los módulos

Una vez que tenga seleccionado un módulo podrá configurar sus parámetros siguiendo los siguientes pasos:

1) Seleccione *"configure"* del menú de opciones. Será llevado a una forma de configuración, donde podrá introducir valores a los parámetros modificables de cada módulo.

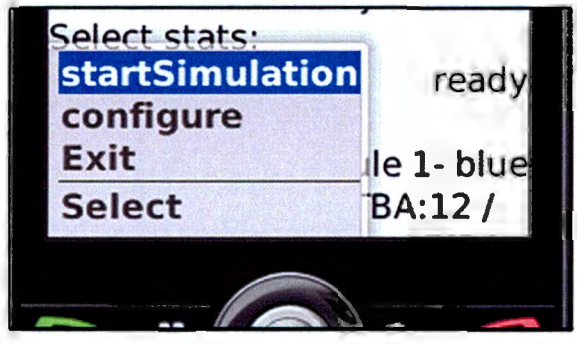
2) Una vez que haya terminado de configurar el módulo deberá seleccionar "OK" del menú de opciones (regresará a la sesión de grupo donde podrá ver los cambios en la lista de módulos). Sí no quiere guardar los cambios de a la configuración

seleccione "*back*" *del menú de opciones* (regresará al menú de sesión de grupo pero no verá ningún cambio en la lista de módulos).



C.2. Ejecución de la Simulación

Una vez que todos los usuarios indiquen que están listos para que comience a ejecutarse la simulación (que tengan al lado de su nombre de usuario lo siguiente: <ready: true>), cualquiera de ellos podrá echar a andar el simulador seleccionando "*start*" del menú de opciones. Sí no están listos todos los usuarios en su status, indicado al lado de su nombre en la lista de usuarios, la simulación se ejecutará.



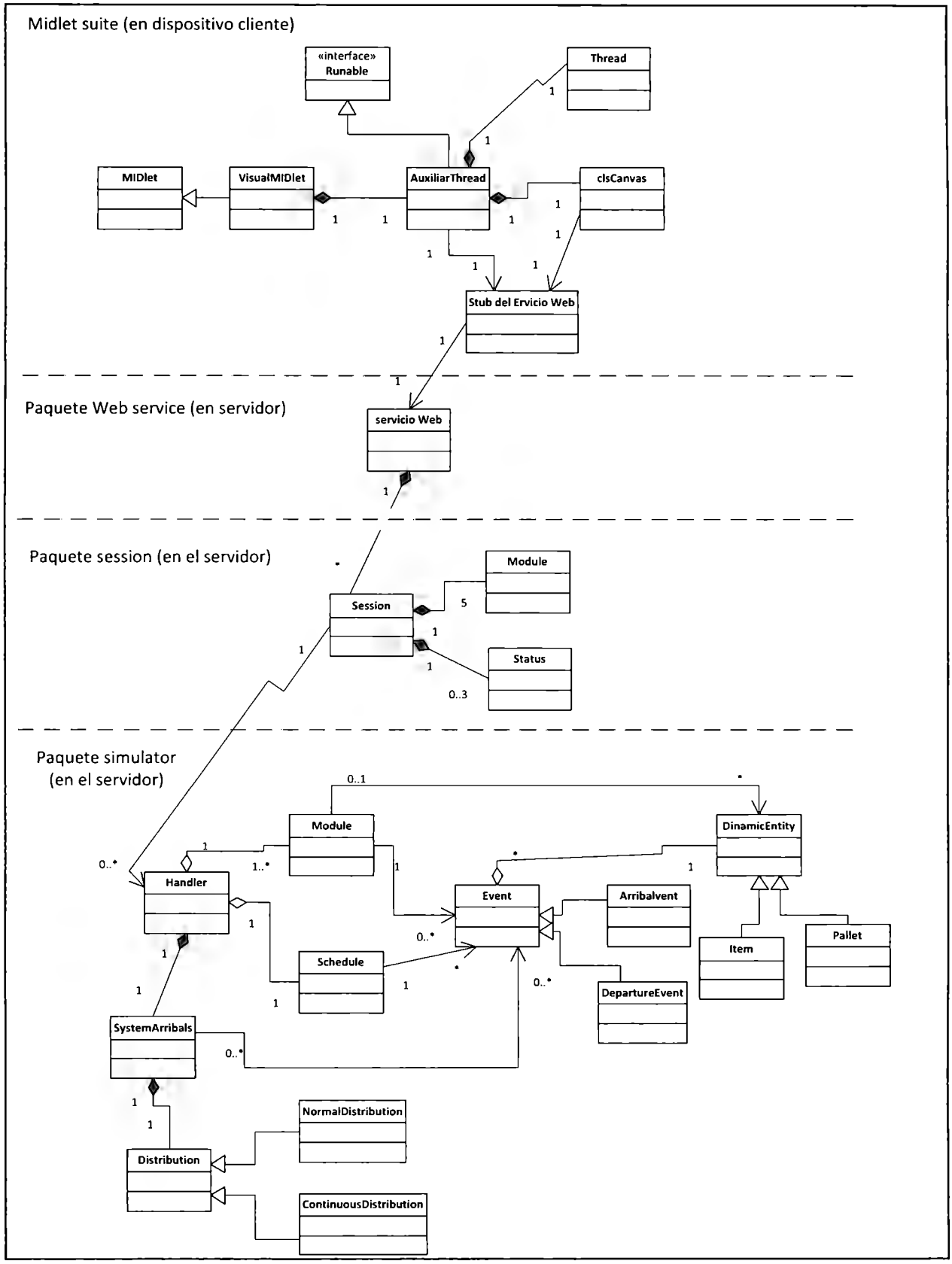
Una vez que se ejecute la simulación los usuarios podrán ver una animación de los eventos generados a partir de esta en sus dispositivos móviles, la cual será en tiempo real, aunque los usuarios podrán acelerar la velocidad de dicha animación apretando las flechas del teclado del dispositivo BlackBerry: derecha para acelerarla, izquierda para desacelerarla.

La animación mostrará las piezas o pallets que se encontrarán en cada módulo a medida que avanza el tiempo. A continuación se especifican los términos empleados.

State: Determina si la simulación se está ejecutando en tiempo real o, a una velocidad mayor o menor

- m1: Módulo 1
- m2: Móulo 2
- m3: Módulo 3
- m4: Módulo 4
- m5: Módulo5

Apéndice B.- Diagrama de Clases



Bibliografía

1) Balci, O 1988, *The implementation of four conceptual frameworks for simulation modeling in high-level languages,* technical report: TR-88-21, Virginia Polytechnic Institute & State University, Blacksburg, VA, USA, pp. 287-295. Consultado en 2009 en ACM Digital Library.
2) Banks, J, Carson, J, Nelson, BL & Nicol, DM 2001, *Discrete-event system simulation.* Third edition. Pretice Hall, USA.
3) Camargos Tavares, AL, Valente, MT, 'A remote display system for java-based mobile applications', *Proceedings of the 2008 ACM symposium on Applied computing*, ACM New York, NY, USA, pp.1918-1922. Consultado en Abril de 2010 en ACM Digital Library.
4) Codl, D, Kazr, J & Koutny, T 2003, 'Comparison-Evaluation of Java-Based Discrete-Time Simulation Tools', *Proceedings of the 37th International Conference MOSIS-2003*, University of West Bohemia, Faculty of Applied Sciences, Czech Republic pp.125-130. Consultado en Septiembre de 2009 en <http://www.j-sim.zcu.cz/Articles/Files/MOSIS2003-CodlKacerKoutny.PDF>.
5) Coulton, P, Rashid, O, Edwards, R & Thompson, R 2005, 'Creating Entertainment Applications for Cellular Phones', *Computers in Entertainment (CIE)*, Vol. 3, Issue 3, pp. 3-3. Consultado en Abril de 2010 en ACM Digital Library.
6) Fischer, M, Mueck, B, Mahajan, K, Kortenjan, M, Laroque, C & Dangelmaier, W 2005, 'Multi-User Support and Motion Planning of Humans and Humans Driven Vehicles in Interactive 3D Material Flow Simulations', *Proceedings of the 37th conference on Winter simulation*, pp. 1921-1930. Consultado en Mayo de 2010 en ACM Digital Library.
7) Fuller, C 1989, 'The Next Generation of Manufacturing Simulation', *Proceedings of the 21st' conference on winter simulation*, ACM New York, NY, USA, pp.840-842. Consultado el 9 de Junio de 2009 en ACM Digital Library.
8) Gartner 2010, *Gartner Says World Wide Mobile Devices Sales Grew 10.8 Percent in Second Quarter of 2010, But Competition Drove Prices Down.* Consultado en agosto de 2010, <http://www.gartner.com/it/page.jsp?id=1421013>

9) Gavalas, D, Economou, D, Kenteris, 'M 2006, The Wireless Internet Technology Landscape', *Proceedings of the First Mobile Computing and Wireless Communication International Conference*, pp. 171-177, Consultado en Abril de 2010 en IEEE Xplore Digital Library.

10) Hall, PS &Anderson, E 2009, 'Operating Systems for Mobile Computing', *Journal of Computing Sciences in Colleges*, vol.25, *Issue* 2, p. 64-71. Consultado en Abril de 2010 en ACM Digital Library.

11) Hamer, C 2007, *Creating Mobile Games/Using Java ME Plataform to Put the Fun into your Mobile Device and Cell Phone*, Apress, E.U.A.

12) Healy, J & Kilgore, RA 1997, Silk: 'A Java-Based Process Simulation Language', *Proceedings of the 1997 Winter Smulation Conference*, pp. 475-482. Consultado el 1 de Diciembre de 2009 en <http://www.informs-sim.org/wsc97papers/0475.PDF>

13) Jacobs, PH & Verbraek, A 2004, 'Single Threaded Specification of Process-Interaction Formalism in Java', *Proceedings of the 36th conference on Winter Simulation*, Winter Simulation Conference, p.1548-1555. Consultado en 2009 en ACM Digital Library.

14) Java-Sim Official 2003, *Tutorial: Working with Java-Sim, consultado en 2009 en <http://sites.google.com/site/jsimoffical/j-sim-tutorial>*

15) Kaar, C 2007, *An introduction to Widgets with particular emphasis on Mobile Widgets* , Mobile Computing technical report number 06/1/0455/009/02, University of Applied Sciences, Hamberg, Austria. Consultado en Abril de 2010 en <http://symbianresources.com/tutorials/techreports/widgets/kaar07widgets.pdf>

16) Kenteris, M, Gavalas, D & Ecnomu, D 2009, 'An Innovative Mobile Electronic Touristic Guide application', *Personal Ubiquitus Computing*, vol. 13, Issue 2, pag. 103-118, Consulatado en Abril de 2009 en ACM Digital Library.

17) Khan, BH 2005 *Managing E-Learning Strategies: Design, Delivery, Implementation and Evaluation*, Information Science Publishing, USA.

18) Kiviat, PJ 1967. *Digital Computer Simulation: Modeling Concepts*, RAND Memorandum RM-5378-PR, Rand Corp, Santa Monica Ca. Consultado en Agosto de 2009 en <http://www.randproject.org/pubs/research_memoranda/2006/RM5378.pdf >

19) Knudsen, J 2008, *Kicking Butt with MIDP and MSA: Creating Great Mobile Applications*, Prentice Hall, E.U.A.

20) Koller, A, Foster, G, Wright, M 2009, 'An Innovative Mobile Electronic Tourist Guide Application', *Personal and Ubiquitous Computing*, Vol. 13, Issue 2, p. 103-118. Consultado en Abril de 2010 en ACM Digital Library.

21) Krahl, D 2003. 'Extend: An Intractive Simulation Tool' *Proceedings of the 35th conference on winter simulation: Driving innovation*, Winter Simulation Conference, p.188-196. Consultado el 2 de Abril en ACM Digital Library.

22) L'Ecuyer, P, Meliani, L &Vaucher, J 2002, *SSJ: A Framework for Stochastic Simulation in Java. Proceedings of the 34th Conference on Winter Simulation*, Winter Simulation Conference, pp. 234-242. Consultado el 3 de Abril de 2009 en ACM.

23) L'Ecuyer, P & Bist, E 2005, 'Simulation in Java with SSJ', *Proceedings of the 37th conference on winter simulation*, Winter Simulation Conference, pp. 611-1620. Consultado el 3 de Abril de 2009 en ACM Digital Library.

24) Mahmoud, QH 2005, *Programming the BlackBerry with J2ME*, consultado en Abril de 2009, < http://developers.sun.com/mobility/midp/articles/blackberrydev>

25) Mahmoud, QH 2005(b), *Service-Oriented Architecture (SOA) and Web Services: The road to Enterprise Application Integration (EAI)*, consultado en Mayo de 2009, <http://www.oracle.com/technetwork/articles/javase/soa-142870.html>

26) McLean, C, Jain, S, Riddick, F, Lee, T, Y 2007. 'A simulation Architecture for Manufacturing Interoperability Testing'. *Proceedings of the 2007 summer computer simulation conference*, Society for Computer Simulation International, San diego, CA, USA, pp. 601-608. Consultado el 3 de Abril de 2000 en ACM Digital Library.

27) Mihalic, K 2008, 'Widegetization of Mobile Internet Experience', T*he 2nd International Workshop on Mobile Internet User Experience*, Yahoo! Inc., E.U.A. Consultado el 15 de Mayo de 2010 en <http://wiki.research.nokia.com/images/7/71/mihalic.pdf>

28) Nogueira, TP, Neto, LCL, Rocha, LS & Andrade, RMC 2008, 'An Adaptation of the Collections Framework, reflection and Object Cloning from J2SE to J2ME', *Symposium on Applied Computing / Proceedingd of the 2008 ACM Symposium on Applied Computing*, ACM New York, NY, USA, pp. 246-250. Consultado en Abril de 2010 en ACM Digital Library.

29) Pang, L & Hudson, W 1999, 'The use of simulation in process reengineering education'. *Proceedings of the 31st conference on Winter Simulation*, pp. 1397-1402. Consultado en Junio de 2010 en ACM Digital Library.

30) Patterson, JF 1991, 'Comparing the programming demands of sinle-user and multi-user applications', *Proceedings of the 4th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, pp.87-94. ACM New York, NY USA. Consultado en Mayo de 2010 en ACM Digital Library.

31) Pressman, RS 2005, *Ingeniería del Software: Un enfoque práctico*, sexta edición, traducido del inglés al español por Murrieta Murrieta, JE, Pineda Rojas, E & Campos Olguín, V, McGraw-Hill, México.

32) Rose, A, Salter, R, Kositsyana, N, Plaisant, C, Rubloff, G & Shneiderman, B 2000, 'Simulation based learning environments and the use of learning histories', *Conference of Human Factors in Computing Systems*, New York, NY, USA, pp. 2-3. Consultado en Octubre de 2009. En ACM Digital Library.

33) Savino, SP 2001, 'WAP: wireless application protocol-wireless wave of the future', *Portland Intranational Conference on Management of Engineering and Technology, 2001*, vol. 1, pp. 178-179. Consultado en Abril de 2010 en IEEE Xplore Digital Library.

34) Schulz, SR & Geiger, CD (2005), 'Transitioning Students from Simulation mechanics to simulation as a Process Improvement Tool: A Multimedia Case Study Aproach', *Proceedings of the 37th conference on winter simulation*, Winter Simulation conference, pp. 2314-2321. Consultado en Abril de 2009 en ACM Digital Library.

35) Schulze, T & Schumann, M. (2000), *Language Based Simulation Models as Management Tools for Assembly Lines,* Proceedings of the 32nd Winter Simulation Conference, Winter Simulation Conference, pp.1393-1401. Consultado el 3 de Abril de 2009 en ACM Digital Library.

36) Sheng, H, Siau, K, Fui-Hoon Nah, F 2010, 'Understanding the values of mobile technology in education: a value-focused thinking approach', *ACM SIGMIS Database*, vol. 41, Issue 2, pp. 25-44, consultado en Marzo de 2010 en ACM Digital Library.

37) Singh, D & Lee, H 2009, 'Database design for global patient monitoring applications using WAP', *Proceedings of the 2nd International Conference on Interaction Sciences:*

Information Technology, Culture and Human, ACM New York, NY, USA, pp. 25-31. Consultado en Abril de 2010 en ACM Digital Library.

38) Singh, I, Brydon, S, Murray, G, Ramachandran, V, Violleau, T, Stearns, B 2004, *Designing Web Services with the J2EE 1.4 Platform: JAX-RPC, SOAP, and XML technologies,* Addison-Wesley.

39) Taj, S, Cochran, DS 1998, 'Simulation and Production Planning for Manufacturing Cells', *Proceedings of the 30th conference on winter Simulation*, Winter Simulation Conference, pp. 973-978. Consultado en Abril de 2009 en ACM Digital Library.

40) Tiacci L & Saetta, S 2007, 'Process-Oriented Simulation for Mixed-Model Assembly Lines', *Proceedings of the 2007 summer computer simulation conference*, Society of Computer Simulation International, San Diego CA, USA, pp.1250-1257. Consultado el 2 de Abril en ACM Digital Library.

41) Van der Zee, DJ, Slomp, J 2005, *Simulation and gaming as a support for lean manufacturing systems: a case example from industry, Proceedings of the 37th conference on winter simulation*, p. 2304-2313. Consultado en Abril de 2009 en ACM Digital Library.

42) VanDoren, V 2003, 'Test your control system with simulation', *Control Engineering*, Vol. 50, Issue 9, pp.42-45. Consultado en Abril de 2009 en EBSCO host Academic Search Premier.

43) Verbraek, A 2004, 'Component-based Simulations: the Way Forward?', *Proceedings of the eighteenth workshop on parallel and distributed simulation*, p.141-148. Consultado en Abril de 2009 en IEEE Xplore Digital Library.

44) Wains, S & Mahmood, W 2008, 'Integrating m-learning with e-learning', *Proceedings of the 9th ACM SIGITE conference on Information technology education*, ACM New York, NY, USA, pp. 31-38. Consultado en Junio de 2010 en ACM Digital Library.