UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

APLICACIÓN INFORMÁTICA PARA LA GESTIÓN DE LA DEMANDA EN CADENAS DE SUMINISTRO UTILIZANDO EL HORIZONTE RODANTE Y TÉCNICAS DE LOTIFICACIÓN

TRABAJO FIN DE MASTER

MASTER EN ORGANIZACIÓN INDUSTRIAL

Autor: Pedro Pagán Pallarés

Director: Francisco Campuzano Bolarín

Cartagena, 7 de Octubre de 2019

Índice

APLICACIÓN INFORMÁTICA PARA LA GESTIÓN DE LA DEMANDA EN CADENAS DE SUMINSITRO UTILIZANDO EL HORIZONTE RODANTE Y TÉCNICAS DE LOTIFICACIÓN

1-Introducción

En la era de las tecnologías y de la información en la que vivimos, nos hemos acostumbrado a dejar en manos de algoritmos y procesadores aquellas tareas que siempre han resultado repetitivas y engorrosas. No hay más que pensar en el día a día de cualquier persona y en todo aquello que nuestro teléfono móvil u ordenador personal es capaz de hacer por nosotros, para darnos cuenta de lo mucho que nos facilitan la vida.

En lo que se refiere a la empresa y al sector industrial, es palpable el hecho de que aplicar tecnología supone una ventaja considerable. De hecho, las empresas que hoy dominan el mercado, son aquellas que han sabido aprovechar de la mejor forma posible las tecnologías disponibles.

La reducción de tiempos y costes, es uno de los principales objetivos de cualquier compañía, ya que minimizándolos se maximizan también los beneficios asociados.

(+) Tecnología → (-) Tiempo → (-) Coste → (+) Beneficio

La ecuación es sencilla, aunque la implementación de estrategias que se apoyen en dicha fórmula, no lo es tanto. Es necesario averiguar primero aquellos puntos en los que la aplicación de una nueva técnica, supondrían mejorar de algún modo los procesos productivos.

En este sentido, lo que este trabajo pretende aportar, es un acercamiento a la reducción de tiempos en la toma de decisiones, a través de la simulación de cadenas de suministro. Sabiendo cómo se comportaría el sistema, se podría decidir aplicar unas políticas u otras.

Gracias a la simulación de modelos, se podrán experimentar cuáles serán las respuestas de un sistema logístico ante distintos estímulos aplicados al mismo.

La aplicación de técnicas de simulación para la exploración de estrategias no es algo novedoso, pero, sin embargo, el uso de la tecnología y más concretamente, de la potencia computacional, hace que se puedan explorar muchas más y de forma mucho más rápida.

Existen programas que facilitan las labores, tanto de modelado de sistemas, como de generación de algoritmos de iteración, permitiendo así elaborar interfaces intuitivas, con las que se puede construir un sistema de simulación muy completo y veloz.

2-Objetivos

El gran objetivo de este trabajo es desarrollar una aplicación informática orientada a la planificación de la producción en cadenas de suministro creadas con Dinámica de Sistemas (DS) utilizando técnicas de horizonte rodante. Las órdenes de reabastecimiento se generan con algoritmos de lotificación que obtienen la información de la previsión que suministra el Horizonte Rodante (HR).

A continuación, se exponen cuáles son los subjetivos que se persiguen con la elaboración de este trabajo.

- **Ilustrar la utilidad de la simulación en el proceso de toma de decisiones:** en el trabajo, se pretende demostrar en líneas generales, cómo es posible agilizar la toma de decisiones en una empresa, concretamente en el área de la logística a través del uso de software de simulación.

  Cuando se define el concepto de cadena de suministro, es inevitable hablar sobre las dificultades que entraña el poder ajustarla de modo que todo quede bien engranado. El hecho de que sean muchos actores y muchas variables, hace que el sistema sea difícil de controlar tanto como se desearía.

A veces, cuando se estudian las causas de un comportamiento ocurrido en la cadena, se pueden llegar a extraer conclusiones importantes, sin embargo, en estos casos ya se habrá llegado tarde para actuar.

Es por ello, que el uso de las predicciones y simulaciones, permite adelantarse a los acontecimientos, tomando así decisiones de forma anticipada y de forma más eficiente.

- **Guiar al lector en la elaboración de sus propios sistemas de simulación:** el objetivo específico del trabajo es generar un tutorial que sirva de referencia para aquellas personas que estén interesadas en construir un sistema de simulación, adaptado a sus necesidades.

  En este punto se podrían identificar **dos tipos de usuarios finales** para este tipo de aplicaciones, en función del uso que se quiera hacer de ellas:

  - Uso empresarial: serían aquellas personas que se van a valer de los resultados de las simulaciones para poder extraer conclusiones a nivel estratégico en la toma de decisiones de su empresa. Podríamos decir que, en última instancia, este usuario pretendería minimizar pérdidas en su organización (Tiempos, Costes, Etc.).

  - Uso didáctico: aplicar este tipo de sistemas puede resultar muy interesante para el público más académico, que busca complementar ciertos estudios en un área con técnicas avanzadas de simulación.

    Poder experimentar con un sistema sometiéndolo a distintas entradas, puede ser de gran ayuda en aquellos casos en los que operar con el sistema real, pueda resultar complicado o imposible. (Ej. Estudios poblacionales).

3-Estructura

El trabajo está dividido en cuatro apartados, a continuación, se resume el contenido de cada uno de ellos.

- **Contexto**: en este apartado se describirán los principales conceptos implicados en el desarrollo del trabajo.

  Por un lado, se definirá lo que es **una cadena de suministro**, ya que es importante situar al lector en la problemática asociada a le gestión de recursos en logística. Se hará hincapié en aquellos puntos que sean relevantes en el desarrollo de la aplicación que se quiere construir.

  Se expondrá la metodología utilizada, **dinámica de sistemas y la simulación de modelos**, ya que conocer los conceptos básicos, ayudará a entender mejor el porqué de la utilización de esta técnica en el estudio de la cadena de suministro.

- **Tutorial:** la guía o manual de diseño de la aplicación es el núcleo del trabajo. En él se describirá paso a paso como elaborar un sistema que permita la simulación de sistemas logísticos de forma genérica, es decir, se definirán cada una de las partes necesarias para construirlo. Para ello se enumeran las **herramientas** necesarias para hacerlo, así como los conceptos que debe conocer el usuario que decida utilizarlo.

  El manual está pensado para cualquier usuario que tenga ciertos conocimientos de programación básica, así como de la cadena de suministro. No obstante, dado el grado de generalidad del manual, se podría aplicar a otro tipo de sistemas distintos a una cadena de suministro, haciendo una extrapolación de los conceptos aquí descritos.

- **Ejemplo**: en esta sección se expone un ejemplo de aplicación del software desarrollado, de modo que el lector pueda tener una referencia a la hora de elaborar su propio diseño.

  Además, este ejemplo forma parte del objetivo general de este trabajo, la ilustración de la potencia de esta herramienta en la toma de decisiones. Esto es, se sacarán conclusiones que permitirán que el lector comprenda las ventajas del uso del sistema.

  El ejemplo es:
  Simulación de una cadena de suministro con dos niveles. Un ejemplo básico de aplicación donde existirán un consumidor final de productos, y un fabricante de los mismos, que utiliza predicciones de demanda y demanda real para ordenar pedidos a producción.

  En este ejemplo, se detallarán las fases de realización del sistema, exponiendo los flujogramas, parte del código utilizado en los anexos, de modo que junto a la guía o tutorial genérico, sirvan como punto de partida para la elaboración de nuevos modelos.

- **Conclusión**: se contrastarán los objetivos marcados con el desempeño realizado, viendo los puntos fuertes y los puntos débiles, así como futuras posibles mejoras en la implementación y desarrollo de estas ideas.

4-Contexto

En este punto se hablará de los principales conceptos implicados en el trabajo y bases teóricas en las cuales se apoya el mismo. Realmente y como se ha descrito en otras secciones, el trabajo es una guía para elaborar sistemas de simulación complejos, sin

embargo, está descrito en profundidad la construcción de un sistema logístico como ejemplo para su desarrollo, por lo que se han definido algunos conceptos básicos sobre la cadena de suministro.

4.1-La cadena de suministro

La gestión de la cadena de suministro (GCS) es la integración de los procesos de negocio claves desde el usuario final a través de los proveedores originales que proporcionan productos, servicios e información que añade valor a clientes y otros interesados (Lambert y Cooper, 2000).

Miembros de la Cadena de Suministro: Todas las empresas u organizaciones con quienes la empresa central actúa recíproca, directa o indirectamente a través de sus proveedores o clientes, desde el punto de origen al punto de consumo.

Miembros primarios. Empresas autónomas que llevan a cabo actividades de valor añadido.

Miembros de apoyo. Empresas que proveen los recursos, conocimientos y utilidades para los miembros primarios.

De forma práctica, la cadena de suministro, es el sistema que trata de satisfacer las necesidades de los clientes, sirviéndose del aprovisionamiento del producto o servicio que se requiere, de forma eficiente.

Un sistema logístico es complejo, y en él intervienen un gran número de actores. La complejidad reside en encontrar el equilibrio entre los intereses de cada uno de los participantes del sistema. El cliente, desea satisfacer su necesidad con calidad, a tiempo y al menor precio posible; El suministrador, desea obtener mayor utilidad, minimizando la función de costes asociados al proceso.

En el trabajo se utilizará el concepto de efecto **BullWhip**, o también conocido como efecto látigo. Hace referencia a los grandes desajustes que pueden darse entre la demanda real de los consumidores y la demanda de los actores intermedios que participan en la cadena de suministro, afectando tanto al stock de los puntos de venta como al almacenamiento en los grandes almacenes de los centros de distribución.

La falta de información o la variabilidad en los tiempos de suministro, pueden ser causa de que la demanda varíe considerablemente de unos niveles a otros, provocando así abastecimientos variables y consecuentes aumentos de costes e insatisfacción de la demanda.

Se estudiará como varía este efecto en función de ciertos parámetros de entrada del sistema.

4.2-La dinámica de sistemas y la simulación de modelos

La dinámica de sistemas, es una metodología que permite el análisis de los sistemas, centrándose en las interrelaciones entre las variables así como en las realimentaciones recíprocas entre los mismos.

La dinámica de sistemas, el nuevo término que empleo Forrester(1961) para la previamente denominada dinámica industrial, por la cual estudió las características de la retroalimentación (*feedback*) de la información de la actividad industrial para mostrar cómo la estructura organizativa, la amplificación (en políticas), y los retrasos temporales (en decisiones y acciones) interactúan para influir en el éxito de la empresa.

Dentro de las alternativas para modelar y simular un sistema de aprovisionamiento, la **dinámica de sistemas** proporciona un método que permite una descripción detallada de los sistemas.

Permite profundizar en ciertos aspectos que pueden resultar muy relevantes a la hora de analizar las interrelaciones entre variables, así como examinar las realimentaciones que se producen entre las variables del sistema.

Gracias a DS, se puede ver a las Empresas como sistemas con seis tipos de flujos: materiales, productos, personal, dinero, pedidos e información.

Otro método muy utilizado para el análisis de sistemas, es mediante Hojas de Cálculo (Excel), siendo éste un método mucho más simplificado y normalmente menos realista.

La ventaja principal es que es fácilmente interpretable, pudiendo ser muy útil en ciertos casos en los que el número de variables implicadas es menor. Suele ser un método más apto para su uso por parte de directivos.

En este trabajo se aborda el estudio con dinámica de sistemas ya que se pretende hace un análisis exhaustivo de las relaciones entre los parámetros del sistema.

5-Tutorial genérico

Este punto del trabajo pretende ser en sí mismo un manual de construcción de un sistema de simulación genérico. El objetivo de hacerlo genérico, es que los conceptos aquí descritos puedan ser utilizados como base para el diseño de sistemas de distinta índole.

En el capítulo se harán algunas referencias a modelos de cadenas de suministro, pero la técnica a utilizar podría ser la misma en el caso de aplicarlo a cualquier otro estudio que desee realizar.

Primero se definirá la estructura o arquitectura del sistema. Posteriormente se hará una descripción de las herramientas utilizadas, tanto software como matemáticas. Una vez

explicada la interacción entre los elementos y conocidas las herramientas, se detallarán los pasos que se seguirían a la hora de abordar la construcción de un sistema.

Es importante destacar que no existe una única manera de desarrollar sistemas de modelado y simulación. En función de la experiencia y preferencias del desarrollador, se podría diseñar un sistema equivalente con arquitecturas y softwares distintos.

5.1-Estructura del sistema

En la imagen se puede observar cuales son los elementos principales que conforman la arquitectura, así como la interacción existente entre ellos.

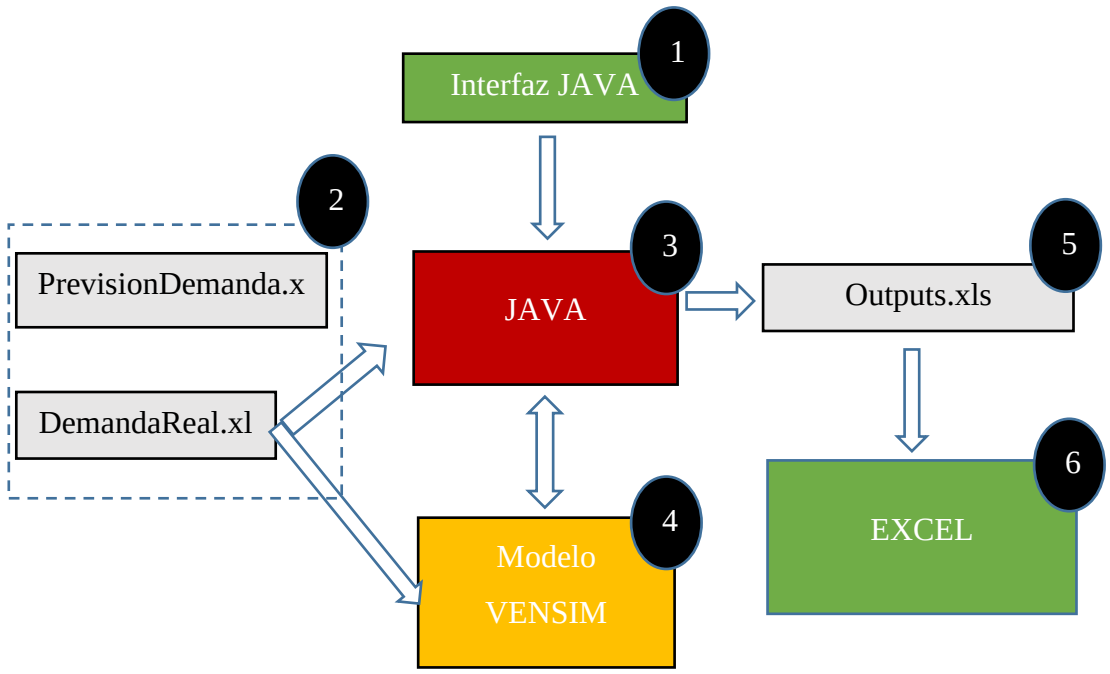


Figura 1 Arquitectura del sistema

Elementos de la arquitectura:

1-**Interfaz de usuario**: pantallas de la aplicación a través de las cuales se introducen ciertos parámetros y configuraciones del programa. Una vez cargados todos los datos, se ejecuta el programa desde aquí.

Lo ideal es que esté diseñado de forma que, al usuario final, le resulte intuitivo y sencilla su utilización, ocultando todo lo posible la complejidad del programa. Una interfaz amigable permite que el usuario se centre en lo realmente esencial de su estudio.

2-**Entrada de datos**: archivos de Excel necesarios para la simulación. A través de una funcionalidad del programa, estos podrán ser modificados desde la Interfaz principal.

En el caso de una cadena de suministro, los datos que sería necesario introducir serían aquellos relacionados con la previsión de demanda y con la demanda real. Esto se detallará más adelante en el apartado Definición del Sistema.

3-**Programa**: encargado de ejecutar las funcionalidades requeridas. Será el que haga de enlace entre VenSim, Excel y el usuario. Diseñado desde Java Eclipse en este caso y siendo el pilar principal del sistema.

En el ejemplo de este trabajo, será aquí donde se lleve a cabo la lotificación, manejo de VenSim, transformación de datos, ejecución del programa principal y exportación de los datos en formato adecuado.

4-**Modelo VenSim de la cadena logística**: fichero VenSim en el cual está representado en diagrama de flujo, la estructura de la cadena de suministro (o sistema cualquiera) que se va a simular. Dependerá del usuario y del modelo que quiera representar con sus

peculiaridades, pero sí que se adelanta ya, que hay que prestar atención a ciertas variables que estarán interconectadas con Java, para poder ser controlado desde el programa.

5-**Salida de datos**: ficheros de Excel con la información de salida de los casos simulados. Son tablas de datos reflejando todos los escenarios que se han generado con los parámetros de entrada. Es el resultado de todas las simulaciones.

Aunque ya en este punto, se tienen todos los datos del resultado, se hace bastante complicado su interpretación debido al gran tamaño de datos que se manejan en estos sistemas.

		Ch=2.0	Ch=2.0	Ch=2.0	Ch=5.39	Ch=5.39	Ch=5.39	Ch=8.0	Ch=8.0	Ch=8.0
CP		Total Costs	Fill Rate	BE	Total Costs	Fill Rate	BE	Total Costs	Fill Rate	BE
500	OUL	1484121,38	82,45	1,63	1543275,00	82,45	1,63	1588818,00	82,45	1,63
500	SM	997685,06	88,25	1,74	1064135,63	88,32	1,66	1121469,50	88,32	1,66
500	WW	989667,63	88,32	1,66	1064135,63	88,32	1,66	1121469,50	88,32	1,66
1000	OUL	1493121,38	82,45	1,63	1552275,00	82,45	1,63	1597818,00	82,45	1,63
1000	SM	989360,75	88,43	2,04	1081144,88	88,25	1,74	1129469,50	88,32	1,66
1000	WW	962279,69	88,63	1,67	1072135,63	88,32	1,66	1129469,50	88,32	1,66
1500	OUL	1502121,38	82,45	1,63	1561275,00	82,45	1,63	1606818,00	82,45	1,63
1500	SM	570565,44	92,34	2,32	1088644,88	88,25	1,74	1147127,13	88,25	1,74
1500	WW	775054,44	89,87	1,92	1080135,63	88,32	1,66	1137469,50	88,32	1,66
2000	OUL	1511121,38	82,45	1,63	1570275,00	82,45	1,63	1615818,00	82,45	1,63
2000	SM	632077,00	94,21	2,24	1052869,88	88,63	1,67	1154627,13	88,25	1,74
2000	WW	805386,38	93,18	2,18	1052869,88	88,63	1,67	1145469,50	88,32	1,66
2500	OUL	1520121,38	82,45	1,63	1579275,00	82,45	1,63	1624818,00	82,45	1,63
2500	SM	409623,91	96,55	2,59	1096526,13	88,43	2,04	1162127,13	88,25	1,74
2500	WW	417534,25	96,43	2,33	1060869,88	88,63	1,67	1153469,50	88,32	1,66
3000	OUL	1529121,38	82,45	1,63	1588275,00	82,45	1,63	1633818,00	82,45	1,63
3000	SM	246364,84	99,04	3,37	666961,25	92,38	1,87	1126297,75	88,63	1,67
3000	WW	252645	98,970001	2,24	1068869,875	88,629997	1,67	1126297,75	88,629997	1,67

Figura 2 Salida de datos del Ejemplo "Cadena de Suministro de 2 niveles"

6-**Excel**: Hoja Excel previamente preparada para la inserción rápida de tablas de salida. A través de Excel, se representan de forma gráfica los distintos escenarios creados para poder sacar conclusiones y tomar decisiones. Es la interfaz de salida. Se ha elegido Excel por su versatilidad y posibilidad de hacer cambios con facilidad.

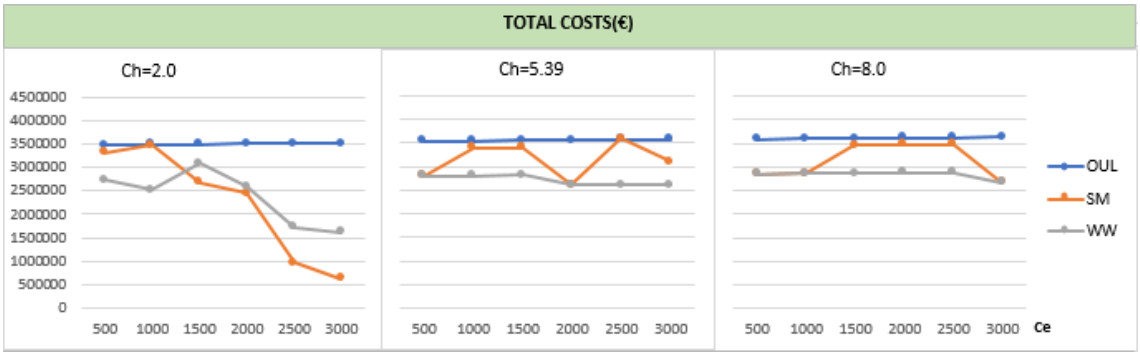


Figura 3 Representación gráfica de resultados "Ejemplo CDS 2 Niveles"

5.2-Herramientas utilizadas

El propósito de este apartado es hacer una descripción de las herramientas y conceptos necesarios para el desarrollo del sistema. Es importante aclarar que los dos primeros puntos (Horizonte Rodante y Algoritmos de Lotificación), son válidos únicamente en el caso de que se esté construyendo un sistema logístico, ya que son conceptos específicos de éste área.

Sin embargo, los siguientes, bien podrían usarse para cualquier desarrollo, indistintamente del área en el que se esté trabajando. En el caso de querer adaptar el diseño para utilizarlo en otra materia, será el diseñador quien deba decidir qué algoritmos o bases teóricas darán cuerpo a su programa.

5.2.1-El horizonte rodante (HR)

El horizonte Rodante es uno de los conceptos centrales del trabajo, ya que la aplicación generará un horizonte rodante utilizando modelos de simulación.

El horizonte rodante, HR en adelante, hace referencia a la actualización continua de la planificación según los nuevos datos de demanda. Esto es, establecido un plan para un determinado periodo de demanda, basado en unos datos de predicción, conforme el tiempo avanza, se recalcula el plan teniendo en cuenta la desviación de la demanda real respecto de la predicción. De este modo, se reajusta continuamente el plan, es una rueda, en la que se inserta continuamente información con la cual se actualiza la estrategia del plan.

Esto se llevará a cabo, generando un escenario variable, que se actualizará para cada simulación conforme llege nueva demanda. Los modelos tradicionales de simulación no son capaces de generar un escenario dinámico que permita actualizar los valores del escenario y simularlo de nuevo, sin embargo, gracias a la aplicación de Java, se puede simular un escenario en cada periodo, actualizando los datos de las previsiones para calcular el lote del periodo siguiente. Esto permite afinar mucho más en la cantidad a solicitar ya que al introducir los datos más nuevos, el sistema cada vez se vuelve más realista.

Por ejemplo, en el caso de querer hacer una planificación para los próximos 12 meses de aprovisionamiento de un determinado bien, una posible forma sería, basarse en la previsión para dichos periodos (en base a datos históricos), y en el primer intervalo adquirir la cantidad que determine la predicción.

Una vez llegue ese periodo, la demanda real respecto de la predicción normalmente, tendrá una diferencia. Es en este momento, cuando se introduce el dato de demanda real en el mecanismo de predicciones haciendo así que la cantidad prevista para el siguiente periodo esté más ajustado a la realidad.

5.2.2-Algoritmos de lotificación SM y WW y técnica OUL

A la hora de elegir los lotes a lanzar en el sistema planteado, se utilizaran dos algoritmos: **Silver-Meal y Wagner-Within**, y una técnica de lotificación, **técnica Order Up To Level,** realizando una comparación entre algoritmo y técnica, para visualizar las ventajas de emplear algoritmos en lotificación.

El uso de la lotificación está encuadrado en la planificación de recursos de la cadena de suministro, entendiéndose esta, como la manera en que se distribuyen los pedidos a lo largo de los periodos siguientes, de forma que minimicen en la medida de lo posible los costes asociados.

¿Cuánto y cuando pedir?

Una buena lotificación tendrá dos implicaciones fundamentales: ahorro en costes y entregas a tiempo a cliente. Dicha lotificación se hará teniendo en cuenta las predicciones de demanda para los periodos posteriores.

Como uno puede imaginar, existen muchas formas y criterios de llevar a cabo la lotificación, pero los fundamentos de todos los algoritmos son los mismos: escoger de entre todas las combinaciones posibles, aquella que mejora una función de utilidad. Normalmente esta función a optimizar estará basada en costes y en algunos casos, con restricciones duras para alcanzar las fechas de entrega.

En este trabajo se han empleado dos algoritmos:

- **Silver-Meal**, en adelante SM.
- **Wagner-Within**, en adelante WW.

SM y WW, son algoritmos iterativos, que testean la función de utilidad para distintas combinaciones de parámetros, detectando aquellos que son “mejores” y seleccionándolos como posibles candidatos para el lote a lanza.

La técnica OUL, está basada en lanzar la orden de pedido cuando se cumplen una serie de condiciones, siendo la cantidad la máxima permitida en almacén. Obviamente, de las tres técnicas, esta última es la más sencilla y la que provocará mayores costes, tanto económicos como de roturas de stock.

En lo que concierne al programa, los algoritmos SM y WW están desarrollados en Java, siendo estos utilizados en cada simulación. Mientras que para testear el modelo con la técnica OUL, se ha generado otro modelo VenSim, que lleva implícito el modelo de aprovisionamiento basado en OUL, es decir, no se hace a través de un algoritmo en Java.

Por tanto, ya se adelanta que, en el modelo construido, habrá dos modelos VenSim, uno en el que la lotificación se hará mediante algoritmos en Java, y otro en el que los lotes son elegidos directamente en cada periodo a través de ecuaciones en VenSim.

5.2.3-VenSim

VenSim es un software de simulación desarrollado por *Ventana Systems.* Principalmente es compatible con la simulación continua, con algunos eventos discretos y capacidades de modelado basadas en agentes. Está disponible comercialmente y como una "Edición de aprendizaje personal" gratuita. (Wikipedia)

Gracias a VenSim, se pueden modelar sistemas de forma intuitiva, buscando relaciones entre las variables que se han considerado en el modelo y escribiendo las ecuaciones de interrelación entre las mismas. Los dos elementos básicos en la programación de un modelo son las variables de flujo y las variables de nivel o acumuladores; a través de ambos se puede representar cualquier modelo.

En el modelo se establece un marco temporal en el que se ejecutará la simulación, así como algunos datos de configuración del modelo (como el tamaño del paso de ejecución), y una vez el sistema es coherente y está bien programado, se inicia la simulación.

Una vez finalizada la simulación, se tienen principalmente dos formas de analizar los resultados, a través de tablas o generando gráficos de las variables que nos interese estudiar.

El procedimiento es sencillo gracias a esta herramienta, la dificultad de un modelado de cualquier sistema radica en la definición del mismo, en la elección de las variables de control, variables de estado, entradas, salidas, ecuaciones, etc. Una vez se tienen claros los conceptos en los que está basado el sistema que se desea modelar, la ejecución a través del programa es relativamente sencilla.

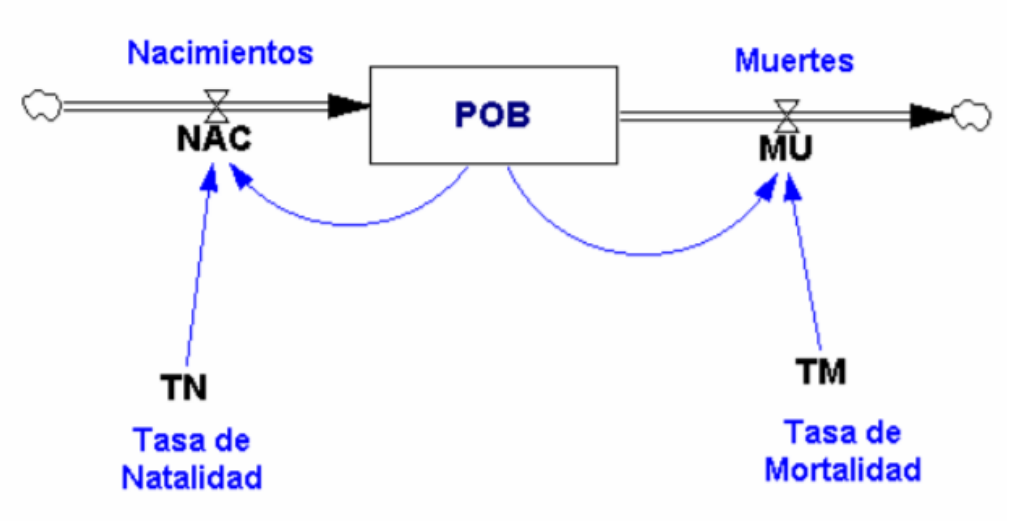


Figura 4 Ejemplo clásico de modelado en VenSim "Estudio poblacional"

En este ejemplo clásico, se muestra cómo funciona el programa VenSim. La población en este caso es la variable de nivel cuyo aumento depende de la diferencia entre los nacimientos(input) y las muertes(outputs). A su misma vez, los nacimientos dependen de la tasa de natalidad y las muertes de la tasa de mortalidad. En función de cómo varíen las

tasas de natalidad en el tiempo, esto tendrá consecuencias en la población, haciendo que aumente o disminuya el nivel.

Ecuaciones del sistema:

t→tiempo (periodo actual)

Población(t)=Nacimientos(t)-Muertes(t)

Nacimientos(t)=Tasa_de_Natalidad*t

Muertes(t)=Tasa_de_Mortalidad*t

El mismo principio se puede seguir con la cadena de suministro, siendo la principal variable de nivel, **el almacén**. Ya sea de productos terminados, semi-terminados, materias primas, etc. De modo que variando entradas y salidas del sistema se modifican los niveles de stock y los costes/beneficios asociados.

5.2.4-Java Eclipse

Figura 5 Pantalla de inicio de Eclipse "Oxygen"

Para conformar el esqueleto y funcionalidades de esta aplicación, se ha elegido el entorno de programación de Java Eclipse, ya que VenSim provee una serie de librerías que hacen posible la interacción entre ambos programas.

De este modo se puede controlar el programa VenSim desde la programación en Java, esto es, dado un modelo dibujado en VenSim coherente, se puede actuar sobre él a través de comandos enviados desde una aplicación desarrollada en Java, de igual forma que haría un usuario en el programa.

Java, como otros entornos de programación, permite mucha libertad a la hora de configurar la estructura del programa además de basarse en un lenguaje de programación potente que permite mucha versatilidad en el código.

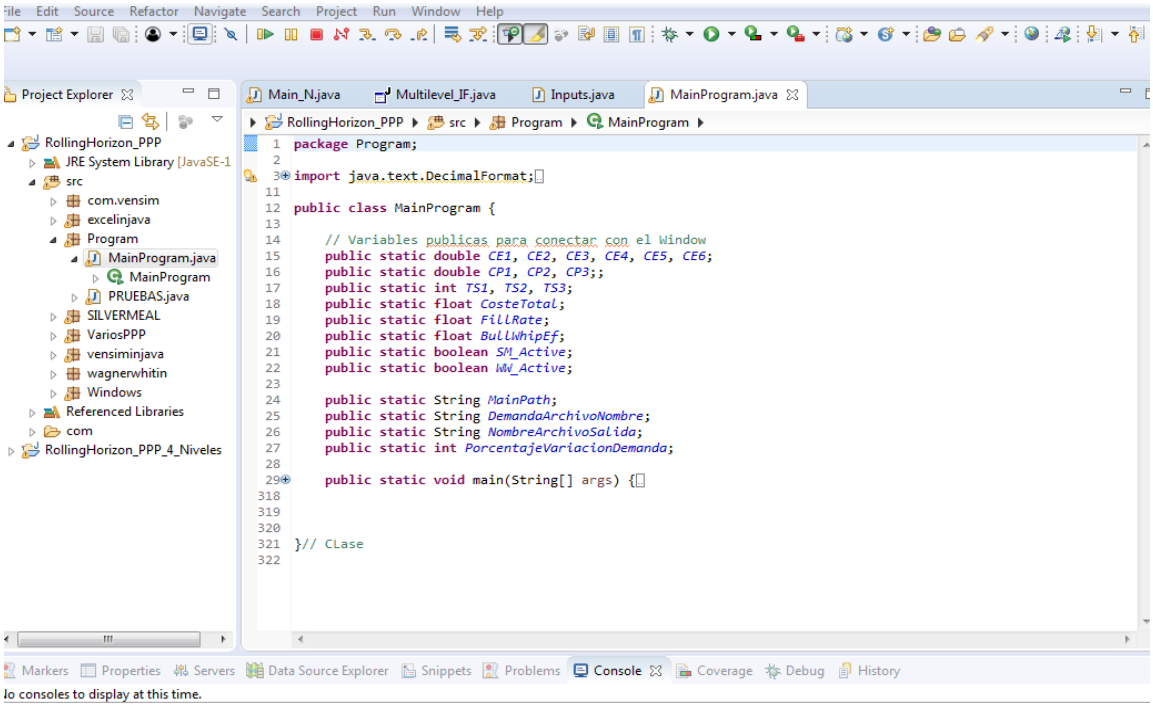


Figura 6 Interfaz de Java Eclipse

Una de las razones por las que se ha elegido Java Eclipse, es que dispone de herramientas para elaborar la interfaz de forma integrada. En la red existen multitud de librerías

disponibles para el público que proporcionan los recursos necesarios para crear aplicaciones gráficas de forma muy intuitiva.
En este caso se ha utilizado *Eclipse WindowsBuilder* para la interfaz. Con un plug-in de este tipo, el código se reduce mucho y el desarrollador se puede centrar en la parte funcional del programa.

La versión utilizada de eclipse en este proyecto ha sido:

Eclipse IDE for Eclipse Committers
Version: 2019-06 (4.12.0)
Build id: 20190614-1200

5.2.5-Excel

Para el manejo de datos tanto en la importación como exportación al programa, se ha elegido Excel debido a que es un programa extensamente conocido y dispone de librerías que lo hacen compatible con los softwares utilizados.

Es una excelente herramienta para organizar los datos de entrada, así como para las exportaciones de los resultados de las simulaciones. Además de por lo comentado anteriormente, Excel ha permitido una realizar una representación gráfica de los resultados, fácilmente adaptable a cualquier modelo.

Es cierto, que se podría haber incluido en el programa principal (Java), una representación gráfica, sin embargo, para este proyecto se ha elegido Excel por su simplicidad a la hora de generar gráficos y ser una herramienta más conocida.

Se recomienda el uso de las versiones actualizadas de Excel, ya que versiones anteriores podrían no ser compatibles con los softwares utilizados. Para este proyecto se ha utilizado Excel 2013.

5.3- Pasos

A continuación, se describen los pasos generales a seguir para construir una aplicación. Para ello, se utilizarán ejemplos sencillos, así como también se hará referencia en algunos puntos al ejemplo después descrito, de modo que el usuario pueda ir entendiendo completamente lo que se ha querido hacer.

Se trata de una serie de pasos conceptuales, que deberán apoyarse con conocimientos técnicos y teóricos del sistema que se pretenda simular. En el ejemplo se mostrará de forma más explícita como realizar cada una de las tareas pertinentes para la elaboración del sistema.

5.3.1- Paso 1: Definición del sistema

Lo principal a la hora de elaborar un sistema de simulación, es plantearse bien qué es lo que se quiere hacer. Parece algo trivial, pero es importante tener claro y acotado el problema para desarrollarlo de forma limpia y clara.

Una buena forma de comenzar con el análisis, podría ser respondernos a algunas preguntas:

1- ¿Qué sistema quiero simular?
2- ¿Qué pretendo conseguir con el análisis?
3- ¿Qué variables están implicadas? ¿Son todas realmente significativas? ¿Son todas medibles?
4- ¿Puedo saber qué relación existe entre cada una de las partes implicadas del sistema? ¿Conozco las ecuaciones?
5- ¿Cuánto podría simplificar el sistema sin perder realismo en los resultados?

En el ejemplo del trabajo "CDS dos niveles", algunas de las respuestas a estas preguntas serían:

1- Una cadena de suministro con dos niveles.
2- Saber que distribuciones de costes (de emisión y de almacenamiento) y tiempos de suministro, proporcionan mejores resultados en términos de nivel de servicio, costes totales y efecto BullWhip.
3- Las variables implicadas son todas aquellas relacionadas con el almacenamiento de productos, pedidos pendientes, capacidad productiva, demanda prevista, etc. Puedo medirlas o calcularlas a partir de otras.
4- Puedo saber las ecuaciones que rigen el comportamiento del sistema a través de la teoría general de la logística.
5- Se podría, por ejemplo, obviar el modo en el que el almacén es capaz de obtener productos terminados, reduciéndolo a una tasa. No perdería calidad en el sistema, ni afectaría a los objetivos del estudio.

Responder a estas preguntas es crítico, ya que será la base a partir de la cual se conformará la estructura del sistema. Tal vez, planteándolas o contestándolas, se obtengan conclusiones que hagan que mejore la calidad o velocidad de realización del análisis.

En este punto, el diseñador debe llegar a la conclusión final, de qué es lo que realmente se quiere, de cuál es su modelo y como construirlo.

5.3.2- Paso 2: Modelado en VenSim

Ya teniendo claro cuál es el objetivo que se persigue, se está en disposición de modelarlo. Para ello, se puede utilizar la herramienta VenSim. El dibujado del flujo es muy intuitivo, se recomienda dibujar primero en un papel los flujos y variables de nivel del sistema para posteriormente generar el diseño en el programa.

Para esclarecer este punto, se realizará un sencillo ejemplo de la realización del modelado para un pequeño sistema logístico, donde los productos son abastecidos por un proveedor con salida ilimitada de producto.

Se aconseja en este apartado, ver diseños ya realizados ya que esto puede llevar a una mayor comprensión del funcionamiento del programa. En la bibliografía se deja un enlace con una web con contenido muy interesante al respecto.

1-Dibujo/boceto del sistema

Se establecen las relaciones principales entre los elementos que se considere imprescindibles del sistema. En este ejemplo básico se puede observar como se ha de satisfacer una demanda a través un almacén de producto, que depende del abastecimiento de los mismos por parte del proveedor, existiendo un tiempo de suministro desde que se manda el producto hasta que se recibe.

Además, las ordenes de reabastecimiento dependerán de la cantidad de producto que haya en el almacén, y en este ejemplo sencillo, la cantidad a pedir es fija.

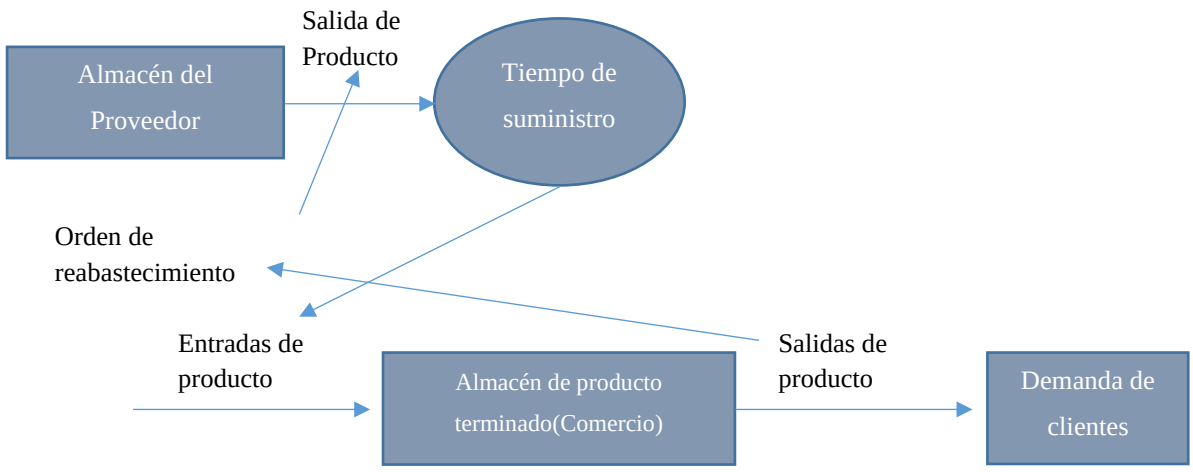


Figura 7 Ejemplo básico de modelado: boceto

2-Modelo en VenSim

Una vez hecho el boceto, se deben decidir cuáles son las variables de nivel y cuales las variables de flujo.

Las **variables de nivel** suponen la acumulación en el tiempo de cierta magnitud, se puede decir que son las variables de estado ya que los valores que toman determinan la situación en la que se encuentra el mismo. Las **variables de flujo** expresan de manera explícita la variación por unidad de tiempo de los niveles.

En el ejemplo, las variables de Nivel serán, el almacén de producto terminado y el almacén de proveedor, el resto son variables de flujo, como las entradas al almacén, o las salidas de producto hacia el cliente.

Ahora se expondrá como se haría el modelo VenSim del pequeño ejemplo del apartado anterior.

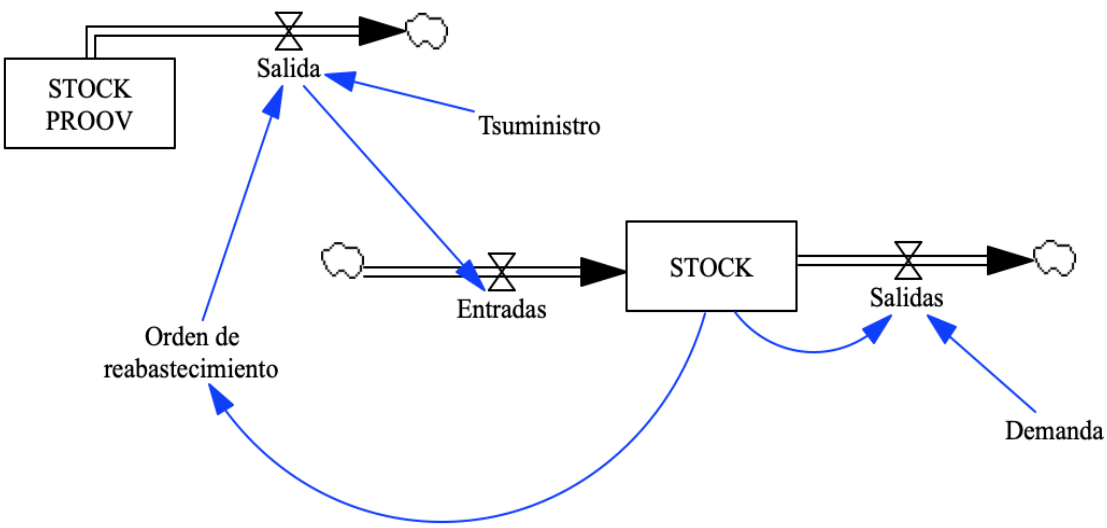


Figura 8 Modelo VenSim de ejemplo básico de cadena de Suministro

Como se observa en la figura anterior, el dibujo del diagrama de flujo en VenSim es prácticamente idéntico al realizado en el boceto, aplicando los conceptos de variables de nivel y de flujo.

Una vez dibujado, es necesario introducir la ecuación del comportamiento de cada una de las variables. Por eso, las variables aparecen sombreadas la primera vez, a la espera de insertar su función. Por defecto las variables de nivel son la diferencia entre la salida y la entrada, quedando solo por establecer un valor inicial de las mismas.

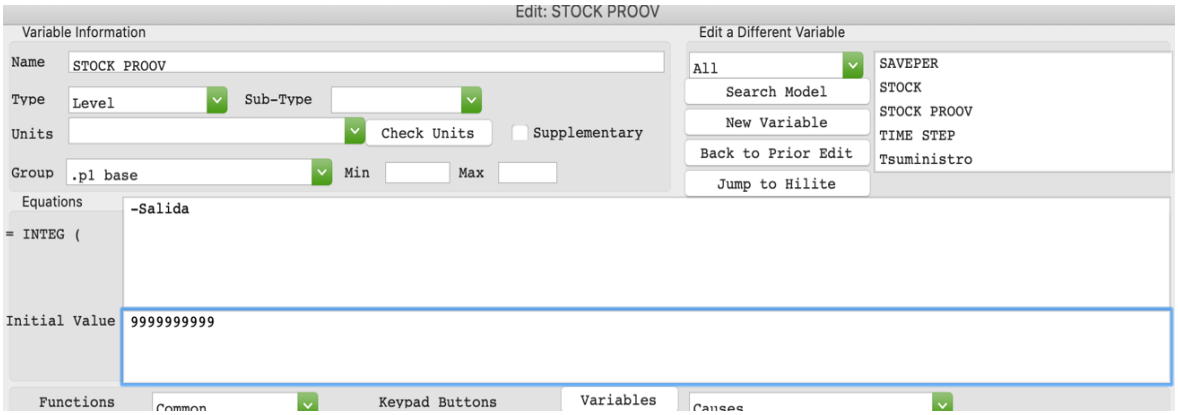


Figura 9 Ecuación de variable de nivel "Almacén de Proveedor"

En la variable de nivel "Almacén de proveedor", al no haber entrada, la ecuación es únicamente la salida. Se ha puesto un número grande en el valor inicial para que haya stock "ilimitado" y siempre se puedan abastecer las órdenes del cliente.

- *Ecuaciones*:
  - **Salida**=DELAY FIXED(Orden de reabastecimiento,Tsuministro,0)
    Comentario: La salida del proveedor es igual a la orden de reaprovisionamiento lanzada por el nivel inferior (en este caso el almacén del comerciante), teniendo en cuenta el retardo provocado por el tiempo de suministro.

  - **Orden de reaprovisionamiento**=IF THEN ELSE (STOCK<0,15,0)
    Comentario: si el STOCK baja de 0 unidades se pide un lote fijo de 15 unidades. Es un caso muy extremo ya que se están provocando roturas de stock.

  - **Tsuministro**= 3.

Comentario: Es una constante. En este caso del número de periodos de retardo es 3 periodos.

- **Entradas**= Salida (del proveedor).
  Comentario: Las entradas de producto, serán igual a la salida del almacén del proveedor.

- **STOCK=**Entradas-Salidas. Valor inicial=20.
  Comentario**:** El almacén será igual a los que entra de producto menos lo que sale, teniendo como valor inicial de stock, 20.

- **Salidas=** MAX (Demanda, STOCK).
  Comentario: la salida del almacén será igual a la demanda, siempre que se disponga de producto en el almacén.

- **Demanda=** INTEGER (RANDOM UNIFORM (3,7,99))
  Comentario: Suponemos en este ejemplo, una demanda aleatoria entre 3 y 7 unidades.

- *Simulación*:

Una vez conformado el sistema, se puede proceder a simularlo. En el caso de que hubiese errores de compilación, el programa daría una salida en la cual podríamos ver dónde están los fallos.

Es el usuario el que decidirá cuantos periodos y de que magnitud serán. Para este ejemplo vamos a simular **50 periodos de 1 mes.** Para representar las gráficas que sean de interés, se seleccionaran las variables que se quieran mostrar, y se seleccionará la opción gráfica.

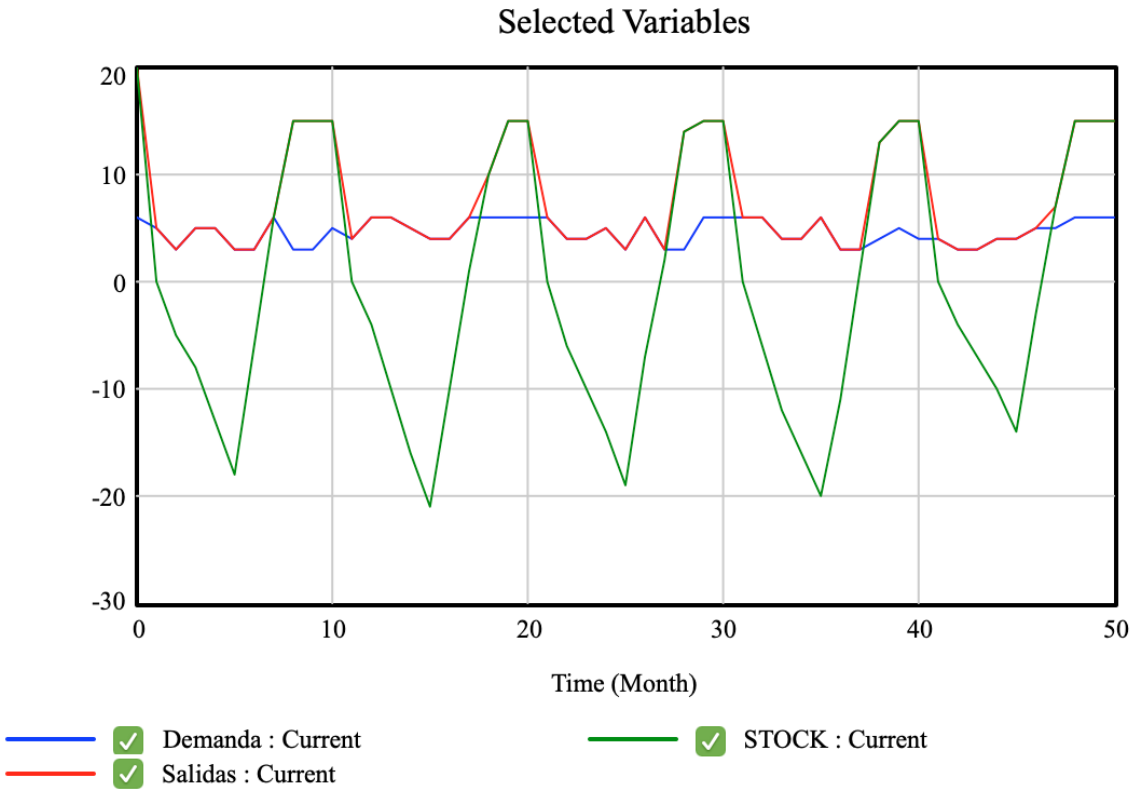


Figura 10 Gráfica resultado de la simulación de la cadena de suministro sencilla

El resultado de las principales variables es el obtenido en la gráfica anterior. Se observa cómo se producen periódicamente roturas de stock, y queda demanda sin satisfacer. Además, la Salida del almacén es justo lo que pide la demanda en cada periodo.

Se trata de un ejemplo muy simple que trata de ilustrar el funcionamiento de VenSim, pero es la base de cualquier cadena de suministro que se quiera modelar.

En los modelos de ejemplo elaborados en los siguientes apartados, se muestra de forma más realista el comportamiento del sistema, existiendo una gran cantidad de variables interrelacionadas. No obstante, tanto en este ejemplo como en los siguientes, se puede identificar como variable de decisión más importante, **la orden** que se manda al proveedor y el momento en que se manda, ya que de ello dependerá la capacidad de respuesta del sistema logístico.

5.3.3- Paso 3: Programa Java

La estructura principal de la aplicación, está programada en Java, ya que este entorno de desarrollo, provee de versatilidad para comunicar todos los softwares o subprogramas implicados en el sistema.

En este apartado se mostrará cómo hacer un programa que interactúe con los principales programas: **VenSim y Excel.** Hechas las conexiones con ambos programas, y teniendo claras las funciones de cada uno de ellos, se podrá elaborar cualquier sistema que se desee, ya sea más complejo o sencillo. El detalle de la programación se podrá ver en el primer apartado de ejemplo del trabajo, donde se mostrarán todas las funciones del programa. En esta sección se expondrá un modelo genérico.

5.3.3.1- Diseño del programa

Para comenzar el diseño, es importante conocer que es lo que queremos que el programa haga por nosotros, y esto se puede resumir en lo siguiente:

TENIENDO UN MODELO DE CADENA DE SUMINISTRO EN VENSIM, SE BUSCA EJECUTAR LA SIMULACIÓN DEL MISMO, DE MODO QUE SE RECORRAN TODOS LOS POSIBLES ESCENARIOS DADOS POR RANGOS DE PARÁMETROS DE ENTRADA.

En la figura siguiente se ilustra como es el bucle que se quiere desarrollar:

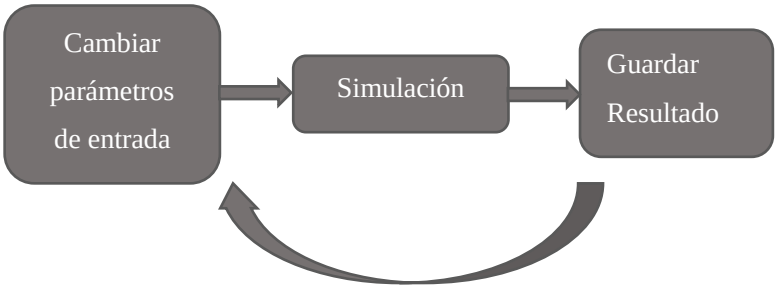


Figura 11 Ciclo de simulación en Java

Como se puede ver en el esquema anterior, se pretende hacer iteraciones del sistema variando los parámetros de entrada, obteniendo así resultados para cada una de las simulaciones. Cada una de las simulaciones a su vez, se ejecutará en VenSim, abarcando los periodos que se hayan predeterminado.

Un ejemplo de lo siguiente, en una cadena de suministro en la que se quiere ver como varía el Coste Total, en función de los costes de emisión y de almacén, sería:

1-Se establecen unos Costes de almacén y costes de Emisión de pedido. (JAVA)
2-Ejecución del programa VenSim con esos parámetros.(JAVA→VENSIM)
3-Guardar Resultados de Coste Total para ese escenario. (JAVA)
4-Vuelve al punto 1 hasta que se acaben las combinaciones de Costes introducidas. (JAVA)
5-Exportación de resultados en una matriz(JAVA→EXCEL)
6-Representación gráfica. (EXCEL)
7-Interpetación: ver para que combinación de costes de almacén y de emisión, los costes totales son mínimos.

- Interacción Java-VenSim:

  Para que el programa funciones correctamente y se pueda controlar el modelo VenSim desde la aplicación de Java, es fundamental conocer cuáles serán las variables que serán modificadas desde cada uno de ellos y cuáles serán devueltas por el modelo como resultados de la exploración.

  En la siguiente figura se muestra cómo debe de ser la interacción entre

ambos programas.

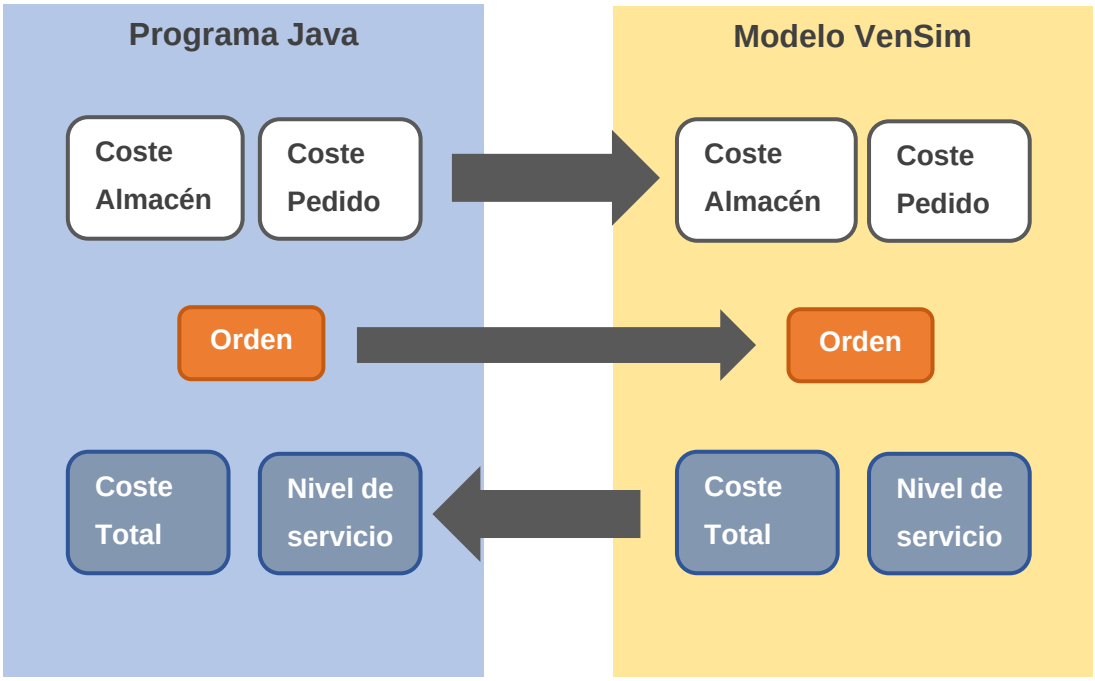


Figura 12 Comunicación Java-VenSim (Conceptual)

En la figura anterior, se puede ver como existen ciertas variables que son entradas del modelo, mientras que otras son salidas del modelo y entradas del programa. Es importante tener claro cuáles serán las variables de entrada/salida ya que éstas existirán en ambos programas.

En el caso de una cadena de suministro, en cada escenario que se quiera modelar, se pasarán al modelo ciertos costes que harán varias la función de costes totales.

Se calculará en **cada periodo** la orden de reaprovisionamiento en el programa Java (mediante lotificación), para pasarla a Vensim y ejecutar la simulación del periodo, devolviendo el modelo, las variables resultado de cada iteración.

El funcionamiento es el mismo que el que llevaría a cabo un usuario de forma manual: se establecerían unos parámetros de entrada, se configuraría la orden de

reaprovisionamiento en base a distintos métodos, se ejecutaría el modelo durante un periodo, y así hasta finalizar la simulación tras n periodos. Una vez finalizada la simulación completa, VenSim transferiría los resultados a Java, y éste los guardará en una matriz, pudiendo comenzarse otra simulación de forma automática.

Como conclusión a este punto, se puede decir que en ambos programas deberán existir:

-**Variables de entrada** que se quieren variar para explorar escenarios (Costes, Tiempos de suministro, etc.).
-**Variable de control** con la que se cambiará el comportamiento del modelo (Orden de reaprovisionamiento).
-**Variables de salida**, que serán los resultados de las simulaciones.

El resto de variables ya estarían configuradas en el modelo, no dependiendo de modificaciones por parte del programa. Además, en el programa Java, las variables deberán apuntar a las variables VenSim haciendo coincidir los nombres con los que están expresadas en el modelo.

5.3.3.2- Diseño de la interfaz

El hecho de elaborar una interfaz, hace que una vez se ha diseñado el programa y el modelo, el usuario de la aplicación pueda limitarse a configurar parámetros y a interpretar resultados.

Es posible también desarrollar un programa cuya interacción con el usuario fuera por barra de comandos, sin embargo, y gracias a las facilidades de los entornos de programación, disponer de una interfaz gráfica facilita mucho el trabajo al usuario, así como le oculta procesos que no son necesarios para trabajar con la aplicación.

Para el diseño de la aplicación, bastaría con disponer de campos de entrada para los rangos de parámetros que se quieren examinar, así como de una ruta donde serán alojados tanto ficheros de entrada/salida como el modelo VenSim.

El desarrollo de este tipo de interfaces con los Plug-in disponibles en Java Eclipse, es muy sencillo, ya que todo el diseño se hace de forma gráfica, teniendo sólo que crear variables asociadas a cada uno de los elementos de la interfaz para interconectarlos con las variables del programa.

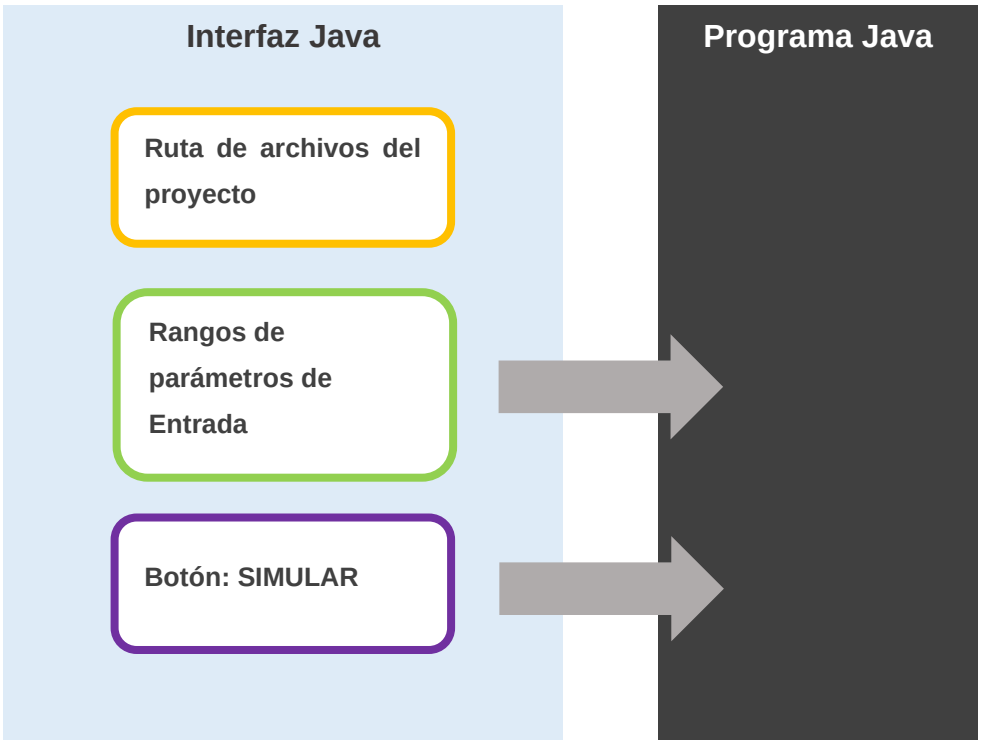


Figura 13 Conexión entre interfaz y programa (Conceptual)

5.3.4- Paso 4: Introducción de hojas de datos

En este apartado se hace referencia a la introducción de datos al programa, aquellos datos de los que se valdrá el algoritmo para llevar a cabo sus cálculos.

En el sistema que se pretende simular y probar, al tratarse de una cadena de suministro, se habrán de introducir los datos correspondientes a la previsión de demanda con la que se hará la lotificación de los pedidos.

Los datos, lo más normal es que tengan un formato de tabla y estén alojados en hojas Excel, de lo contrario, existen formas de convertirlo a formatos legibles por Excel, para de esta forma estandarizar lo máximo posible la manera de trabajar.

Mediante el entorno de desarrollo Java y a través de librerías existentes, se podrá realizar la interconexión entre los datos requeridos y el programa. La idea principalmente es, que en el programa Java desarrollado, **exista un objeto equivalente a la tabla de datos de Excel**, de modo que pueda ser accesible en todo momento, como si se estuviera trabajando directamente con ella.

De forma similar a como controla Java a VenSim, se generarán objetos con los que se apuntarán a las hojas de Excel donde se dispone de los datos que se van a utilizar y de esta forma se trabajará con ellos directamente en el programa.

En el caso concreto del Ejemplo 1, se verá como se apunta desde Java a la hoja de previsiones con la que después, y mediante lotificación se formarán las ordenes de pedido.

5.3.5- Paso 5: Simulación e Interpretación

Una vez se tienen los siguientes puntos establecidos,

- Sistema objetivo
- Modelo VenSim del sistema
- Programa Java controlando VenSim y Excel
- Datos en formato tabla para la inserción

Se está en disposición de proceder a la simulación del sistema con los rangos de parámetros de entrada que nos interese modificar, además se debe de elegir bien aquellas variables que serán la realimentación del estudio, ya que será a partir de ellas de donde se saquen las conclusiones de la simulación.

El orden en el que se debe de ejecutar las tareas en este último paso es:

1-**Introducción de rangos de parámetros en la interfaz** previamente diseñada. Estos rangos de parámetros son los que se recorrerán, y Java ejecutará las simulaciones tantas veces como combinaciones entre parámetros de entrada existan.

2-Establecer la **ruta donde se alojarán los archivos del programa** (Excel de entrada utilizados, Excel con los resultados del programa y modelos de VenSim que serán abiertos durante la ejecución).

3-Indicar en la interfaz el **nombre de los archivos Excel** de entrada que se utilizarán en la ejecución.

4-**Simular**: si todos los datos son coherentes, el programa comenzará la simulación. La duración de la simulación dependerá de la cantidad de combinaciones de parámetros de entrada existentes y para los cuales se generarán escenarios diferentes y sus correspondientes simulaciones en VenSim.

5-**Traspaso de la tabla de resultados**: los resultados de las simulaciones aparecerán alojados en un libro Excel, y en función de como haya sido diseñado, es posible que estén en varias hojas. Estos datos, contienen todos los resultados para cada una de las combinaciones organizados por filas y columnas según se haya configurado en el programa Java.

Debido a la complejidad de interpretación de los resultados en bruto, es aconsejable traspasar estos datos a un nuevo libro de Excel que se tenga previamente configurado, en el que existan gráficas preparadas para ilustrar los resultados.

El formato del libro dependerá del diseño y de lo que se quiera monitorizar, pero la idea es que se pueda copiar una tabla de resultados desde el Excel salida del programa, y pegarla en el libro pre configurado, de modo que automáticamente los resultados aparezcan de forma gráfica y sean más fáciles de interpretar.

6-Ejemplo de uso

6.1- Cadena de suministro con dos niveles

6.1.1-Sistema

El sistema que se va a simular en este ejemplo, se trata de un modelo de cadena de suministro con dos niveles, donde existe un nivel que es el comerciante del producto terminado, y otro nivel que es el fabricante o distribuidor, que abastece al comerciante. A su vez el comerciante debe de satisfacer la demanda existente, en base a predicciones.

La finalidad de simular este modelo, es doble: por un lado, serviría como herramienta para la toma de decisiones en una empresa que se adecuase a la estructura, y por otro, tiene un carácter didáctico, ya que trata algunos conceptos que van mas allá de la toma de decisiones estratégicas, como es el análisis del comportamiento de la demanda en cada nivel (Efecto BullWhip).

Parámetros y variables

En el ejemplo, ser hará variar tres parámetros de entrada:

- **Costes de emisión de pedido:** es el coste asociado a lanzar un pedido.

- **Costes de posesión de producto terminado:** coste de mantener una unidad de producto en inventario. Este concepto de coste se podría desglosar en el coste de almacén y coste de capital (o de oportunidad).
- **Tiempos de suministro:** tiempo existente desde que se lanza la orden al proveedor hasta que se recibe.

Y se obtendrán como salidas, los parámetros:

- **Coste total de la gestión:** serán los costes totales procedentes de sumar cada uno de los costes en cada periodo. En modelos de cadena de suministro, la función de costes suele ser la función de utilidad a minimizar, ya que de ella se deriva directamente, una maximización de los beneficios totales.

- **Nivel de servicio dado al cliente**: número de pedidos satisfechos al final de la simulación. Es decir, al final de los 20 periodos, se hará la cuenta de la demanda real que ha habido, y aquella que se ha podido cumplir.

  Es importante destacar en este punto, que, para este parámetro, únicamente se tiene en cuenta qué porcentaje de demanda ha sido cubierta, **aunque no se haya hecho a tiempo**, cosa que en la realidad es una restricción bastante importante y se debe de tener en cuenta a la hora de minimizar los costes totales.

- **Efecto BullWhip(BE)**, con el que se podrá ver la variabilidad de la demanda en los distintos niveles, provocado en gran parte por los tiempos de suministro y las restricciones en los tamaños de lote.
  El estudio de este parámetro, tiene más que ver con la comprensión del sistema a nivel didáctico, ya que en la realidad este parámetro es de sistema, y no de cada nivel. En líneas generales, sirve para saber como afecta la desinformación en las cadenas de suministro, a la previsión de stocks y consecuente aumento en costes.

Número de simulaciones

Se simularán 20 periodos de un mes.

El número de simulaciones totales que se llevarán a cabo, dependerá de las combinaciones posibles entre los parámetros de entrada dados, en este caso:

Costes de emisión=6; [500; 1000; 1500; 2000; 2500; 3000] €/Pedido
Costes de posesión=3; [2; 5.39; 8.0] €/(Unidad*Periodo)
Tiempos de suministro=3; [2; 3; 4] Semanas
Algoritmos de lotificación=3; [SM; WW; OUL]

6*3*3*3=**162 Escenarios diferentes.** Donde cada escenario supondrá la simulación de 20 periodos. Como se ve, es una gran cantidad de datos para tener que introducirla manualmente. Estos resultados se exportarán en una tabla Excel, y serán interpretados a través de una hoja Excel preparada para su inserción y representación gráfica.

6.1.2-Modelo VenSim

El modelo de la figura recoge la dinámica que rige el sistema que se ha querido representar, con las variables y ecuaciones que se han creído necesarias para la correcta interpretación del mismo.

En el modelo esta por un lado representada la cadena de suministro y sus causalidades, y por otro, diversas herramientas necesarias para el cálculo de los parámetros que permitirán un análisis posterior, como el cálculo del coste total, nivel de servicio o efecto BullWhip.

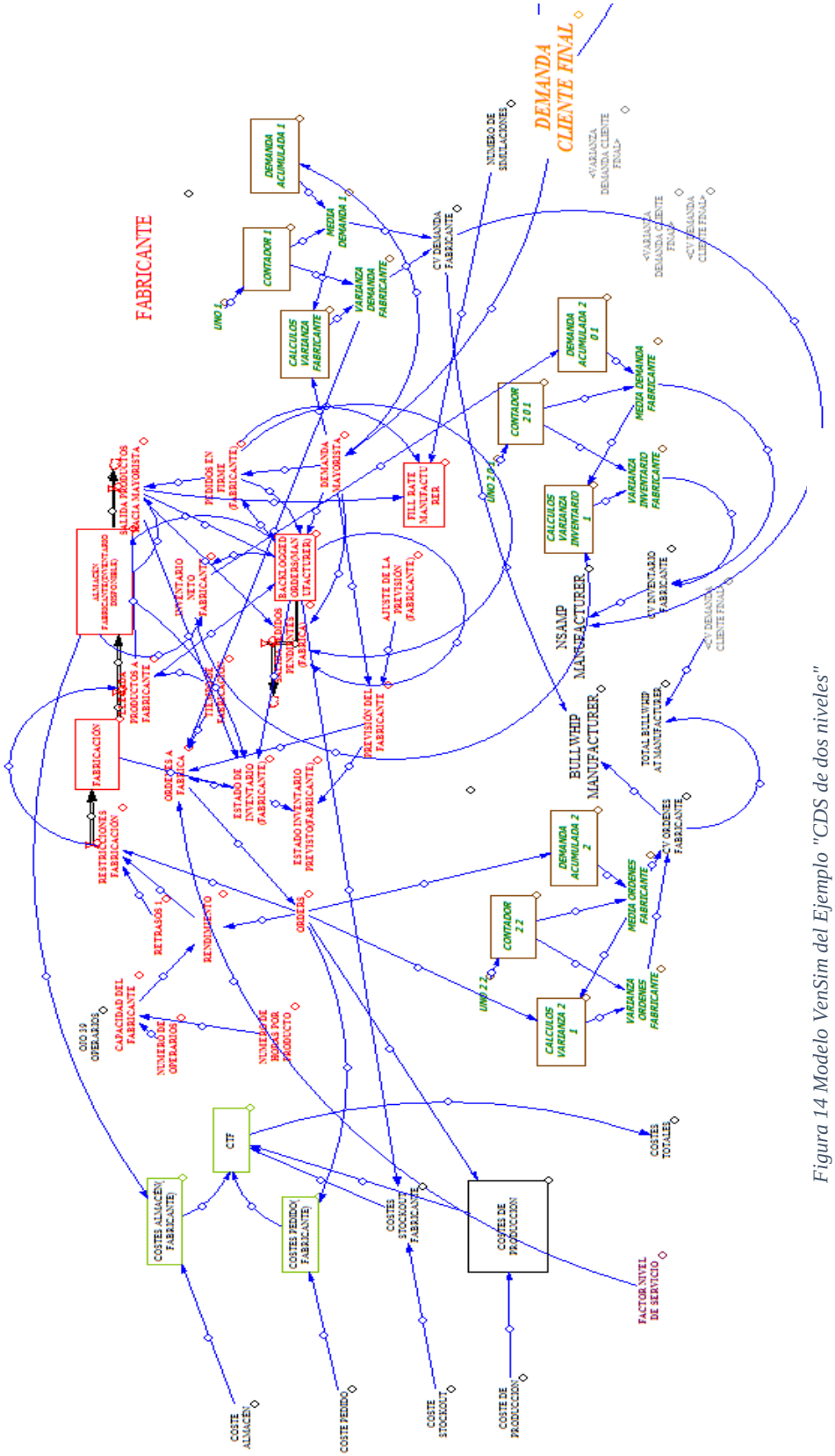


Figura 14 Modelo VenSim del Ejemplo "CDS de dos niveles"

6.1.3-Programa Java

PROGRAMA (Diagrama de flujo)

A continuación, se representan dos flujogramas: el Principal y *GameOn. El Principal contiene el GameOn, pero se ha dividido para hacerlo más legible. Primero se describe el principal y después el GameOn.

***GameOn**, se refiere a la instrucción con la que se ejecuta cada una de la simulación en el modelo VenSim. Con dicha instrucción se simula individualmente cada periodo y se puede considerar un subprograma dentro del hilo principal.

----------PRINCIPAL----------

Nomenclatura del flujograma:

Tsum: Tiempo de suministro
Ce: Coste de emisión.
Ch: Coste de posesión
Tsum.max: número de tiempos de suministro para los que se quiere hacer la simulación. En el ejemplo, Tsum.max=3.
Ce.max: número de costes de emisión para los que se quiere hacer la simulación. En el ejemplo, Ce.max=6.
Ch.max: número de costes de posesión para los que se quiere hacer la simulación. En el ejemplo, Ch.max=3.
Periodos: cada uno de los intervalos de tiempo para los que se realiza una ejecución de la simulación.

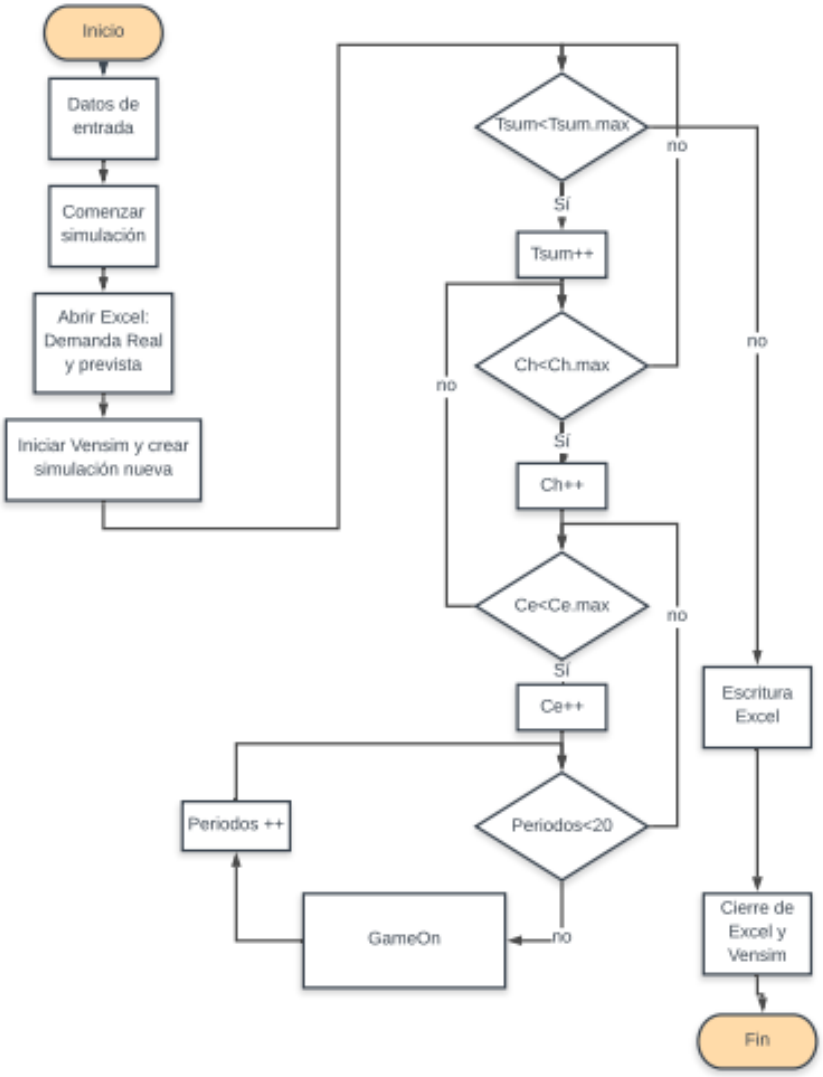


Figura 15 Flujograma principal del programa Java "CDS 2 niveles"

Descripción del proceso:

1-Entrada de datos: Lo primero es la introducción de los datos de entrada en la interfaz gráfica, estos datos son: Tiempos de suministro, Costes de Emisión, Costes de Posesión, Hoja Excel con los datos de previsión de demanda, Hoja Excel con los datos de la demanda real. En la interfaz gráfica se ha de indicar la ruta donde se alojarán tanto los archivos de entrada como los de salida.

2-Inicio de la simulación: una vez están todos los parámetros de entrada, se procede a iniciar la simulación. Si los datos de entrada no fuesen coherentes, saldría un error obligando al usuario a introducir de nuevo los datos de entrada.

3-Abrir de los datos del Excel: el programa toma el control de Excel, abriendo los archivos correspondientes a previsión de demanda y a demanda real. Así mismo y gracias a las funciones de integración Excel-Java, se generan un archivo nuevo en blanco, donde se introducirán posteriormente los datos de salida de la simulación. Por lo tanto, en este punto existen 3 libros de Excel abiertos por Java.

4-Iniciar VenSim y crear una simulación nueva: desde Java se abre VenSim y el archivo .**vpm** del modelo VenSim correspondiente al sistema que se quiere analizar. A partir de este momento, se opera como se haría directamente en VenSim.

Se guarda y se crea una nueva simulación (los parámetros de la simulación pueden estar previamente introducidos o introducirse desde Java). En este caso la configuración de la simulación está hecha en el propio modelo, con lo que no habría que hacer nada más. Ya estaría preparado para ejecutarse.

5-Bucles de exploración de soluciones: a partir de este momento, el programa comienza a generar simulaciones para cada una de las combinaciones de parámetros introducidos. Esto se hace a través de 3 bucles FOR que recorren cada uno de los parámetros combinados con los demás. El último bucle FOR es para recorrer los periodos de la simulación (en este caso 20), ejecutando en cada uno GameOn, que es la simulación correspondiente a cada periodo.

Esta parte es el núcleo del programa, ya que es lo que permite al usuario explorar todas las situaciones sin tener que introducir manualmente cada uno de los casos.

6-Escritura de datos: una vez se tienen todos los resultados en una estructura creada en Java, se copian todos en un libro de Excel, generando una hoja para cada uno de los Tiempos de Suministro.

7-Cierre de programas: una vez acabado el proceso, se cierra tanto Excel como VenSim.

----------GAMEON----------

Nomenclatura del flujograma:

PP: Pedidos pendientes

Prev.Dem: Previsión de demanda

SM: Silver-Meal

WW: Wagner-Within

OUL: Order-up-to-level

GetVar: Función de librería Java de VenSim para obtener variables

SetVar: Función de librería Java de VenSim para asignar valores a variables

Limitador de Lote: Variable del modelo de VenSim

GameOn: Función de VenSim para proceder a ejecutar la simulación de un periodo

FR: Fill Rate, nivel de servicio. Cantidad de pedidos entregados a cliente

TC: Total costs. Costes totales del periodo

BE: Bullwhip effect. Efecto Bullwhip entre los distintos niveles de la cadena

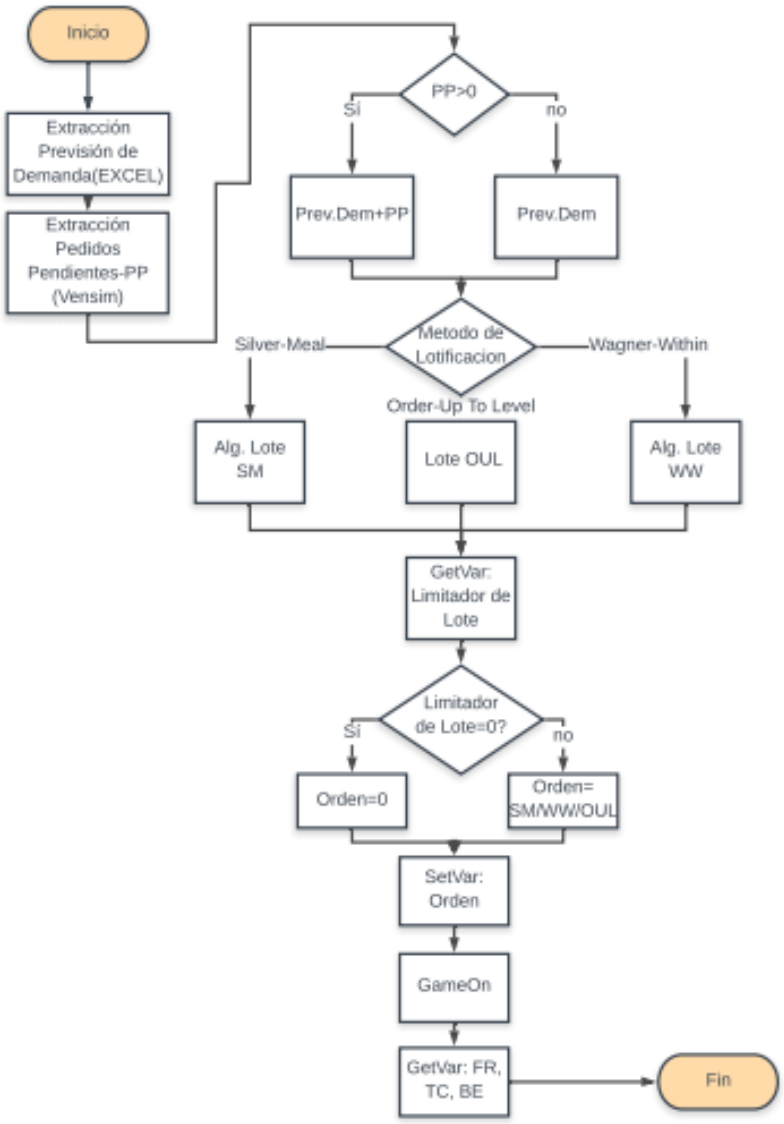


Figura 16 Flujograma "GameOn" del ejemplo "CDS 2 niveles"

Descripción del proceso:

1-Extracción de la previsión de demanda: en cada periodo se extrae la previsión de demanda para cada intervalo.

2-Extracción de Pedidos Pendientes: en cada periodo se extraen los pedidos pendientes existentes en el modelo debidos a roturas de stock en periodos anteriores.

3-PP>0

3.1-SI, Se suman los pedidos pendientes a la previsión de demanda para ese periodo.
3.2-NO, únicamente se tienen en cuenta la previsión para solicitar la orden.

4-Método de lotificación: en cada simulación se ejecutan los 3 métodos, uno detrás de otro. Primero se ejecuta toda la simulación con un método y luego con los restantes, de modo que queda cubierto todo el abanico de posibilidades. Como se ha comentado en apartados anteriores, para ejecutar SM y WW, Java abrirá un modelo de Vensim, mientras que para ejecutar OUL abrirá otro.

4.1 Silver-Meal: se utiliza la función de lotificación de Silver-Meal
4.2 Wagner-Within: se utiliza la función de lotificación de Wagner-Within
4.3 Order-up-to-level: se utiliza el pedido que se obtiene de la ecuación OUL integrada en el modelo Vensim.

5-GetVar: **Limitador de Lote**. Es una variable del modelo que indica si es necesario pedir o no en función de los pedidos pendientes y del estado del inventario. En el caso de ser positiva, esto indica en el programa Java que se debe ejecutar una lotificación y posteriormente enviar una orden.

6-Limitador de Lote =0?

6.1-Orden=0, no es necesario pedir nada.
6.2-Orden=Orden SM/Orden WW/Orden OUL. Será una u otra en función del método que se esté ejecutando en ese momento.

7-SetVar: Orden. En este momento ya sabiendo cual es la orden que se debe solicitar, desde Java (para el caso SM y WW) se envía dicha orden al modelo. En el caso de OUL, la orden se solicitará internamente desde el modelo.

8-GameOn: Se manda una señal desde Java para que Vensim ejecute la simulación de un periodo ya con los datos introducidos. Esta señal se manda en cada periodo.

9-GetVar: FR, TC, BE. Al final de cada periodo se obtienen las respuestas del sistema para los tres parámetros estudiados en este ejemplo. A pesar de que se obtienen para cada periodo, en este ejemplo sólo se exportaran a la tabla final los del último periodo. No obstante, podrían escribirse para todos los periodos en función de lo que se quisiera analizar.

6.1.4-Datos

Los datos utilizados en este ejemplo, son dos tablas de datos:
-Tabla de datos con **previsiones** de demanda
-Tabla de datos con la **demanda real** para cada periodo

En el ejemplo se ha decidido que la demanda real proviniese de un Archivo Excel, de modo que, a través de una función de Java, se pudiese modificar aplicando ciertos porcentajes de aleatoriedad sobre ella, aunque también se podía haber decidido generarla directamente desde VenSim, con las funciones que el mismo programa provee.

En el mismo modelo VenSim, se ha hecho que la demanda real, vaya leyendo periodo tras periodo las filas correspondientes a cada periodo del Excel.

El archivo de Previsión de Demanda, es utilizado en los cálculos de lotificación en cada periodo.

6.1.5- Simulación y resultados

6.1.5.1- Preparación de la simulación y ejecución

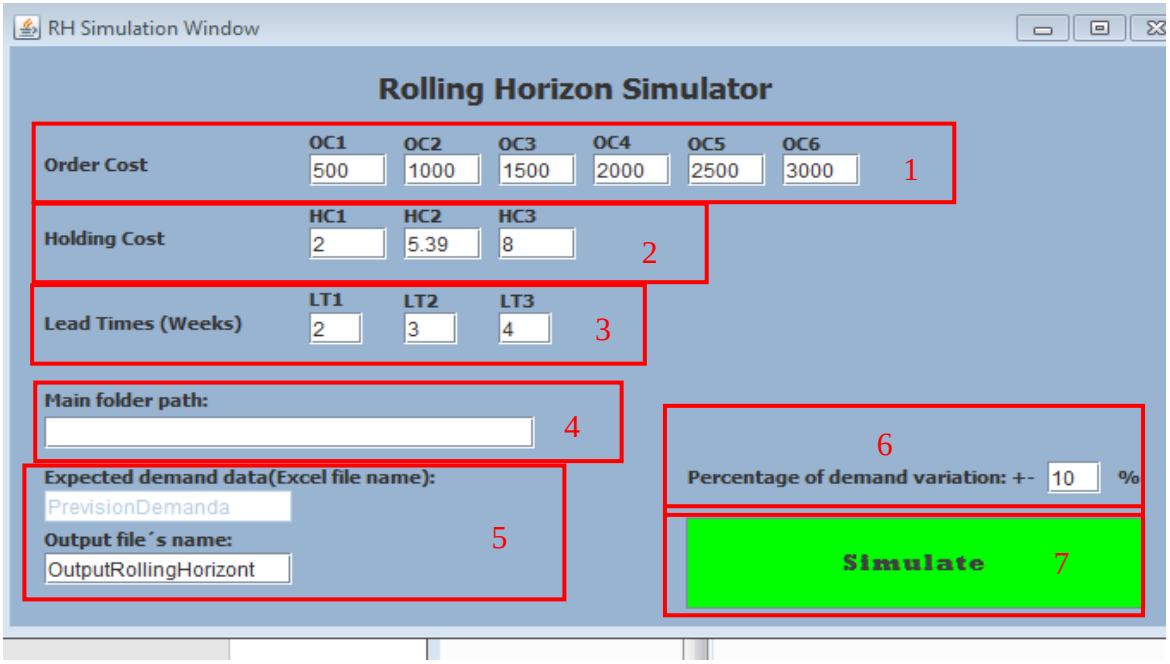


Figura 17 Introducción de parámetros en la interfaz

1-Rango de entrada para los Costes de Emisión (“Order Cost”)

2-Rango de entrada para los Costes de Posesión(“Holding Cost”)

3-Rango de entrada para los Tiempos de Suministro (“Lead Times”)

4-Ruta de la carpeta contenedora del proyecto donde se alojan todos los archivos y ficheros del mismo, así como el modelo VenSim.

5-Hojas de Cálculo

5.1-Previsión de demanda para el cálculo de la lotificación con los algoritmos SM y WW.

5.2-Fichero de salida con las tablas contenedoras de los resultados.

6-Porcentaje de variación: funcionalidad añadida para poder variar desde el programa los datos de previsión de demanda y de demanda real(leída por VenSim), para poder simular distintos escenarios generando previsiones y demandas con porcentajes de aleatoriedad de forma automática sólo con cambiar un parámetro. La función básicamente recorre todas las filas de las tablas de datos, multiplicándolas por un 1+Porcentaje Aleatorio(introducido en la interfaz).

7-Botón para simular.

6.1.5.2-Exportación de los resultados

Con los datos ya en el libro de Excel, organizados en hojas (una hoja para cada tiempo de suministro):

		Ch=2.0	Ch=2.0	Ch=2.0	Ch=5.39	Ch=5.39	Ch=5.39	Ch=8.0	Ch=8.0	Ch=8.0
CP		Total Costs	Fill Rate	BE	Total Costs	Fill Rate	BE	Total Costs	Fill Rate	BE
500	OUL	1484121,375	82,45	1,63	1543275	82,45	1,63	1588818	82,45	1,63
500	SM	997685,0625	88,25	1,74	1064136	88,32	1,66	1121469,5	88,32	1,66
500	WW	989667,625	88,32	1,66	1064136	88,32	1,66	1121469,5	88,32	1,66
1000	OUL	1493121,375	82,45	1,63	1552275	82,45	1,63	1597818	82,45	1,63
1000	SM	989360,75	88,43	2,04	1081145	88,25	1,74	1129469,5	88,32	1,66
1000	WW	962279,6875	88,63	1,67	1072136	88,32	1,66	1129469,5	88,32	1,66
1500	OUL	1502121,375	82,45	1,63	1561275	82,45	1,63	1606818	82,45	1,63
1500	SM	570565,4375	92,34	2,32	1088645	88,25	1,74	1147127,125	88,25	1,74
1500	WW	775054,4375	89,87	1,92	1080136	88,32	1,66	1137469,5	88,32	1,66
2000	OUL	1511121,375	82,45	1,63	1570275	82,45	1,63	1615818	82,45	1,63
2000	SM	632077	94,21	2,24	1052870	88,63	1,67	1154627,125	88,25	1,74
2000	WW	805386,375	93,18	2,18	1052870	88,63	1,67	1145469,5	88,32	1,66
2500	OUL	1520121,375	82,45	1,63	1579275	82,45	1,63	1624818	82,45	1,63
2500	SM	409623,9063	96,55	2,59	1096526	88,43	2,04	1162127,125	88,25	1,74
2500	WW	417534,25	96,43	2,33	1060870	88,63	1,67	1153469,5	88,32	1,66
3000	OUL	1529121,375	82,45	1,63	1588275	82,45	1,63	1633818	82,45	1,63
3000	SM	246364,8438	99,04	3,37	666961,3	92,38	1,87	1126297,75	88,63	1,67
3000	WW	252645	98,97	2,24	1068870	88,63	1,67	1126297,75	88,63	1,67

TS=2 TS=3 TS=4 TSV(2,4) +

Figura 18 Primera hoja (Ts=2) del libro generado por la aplicación con los resultados de la simulación

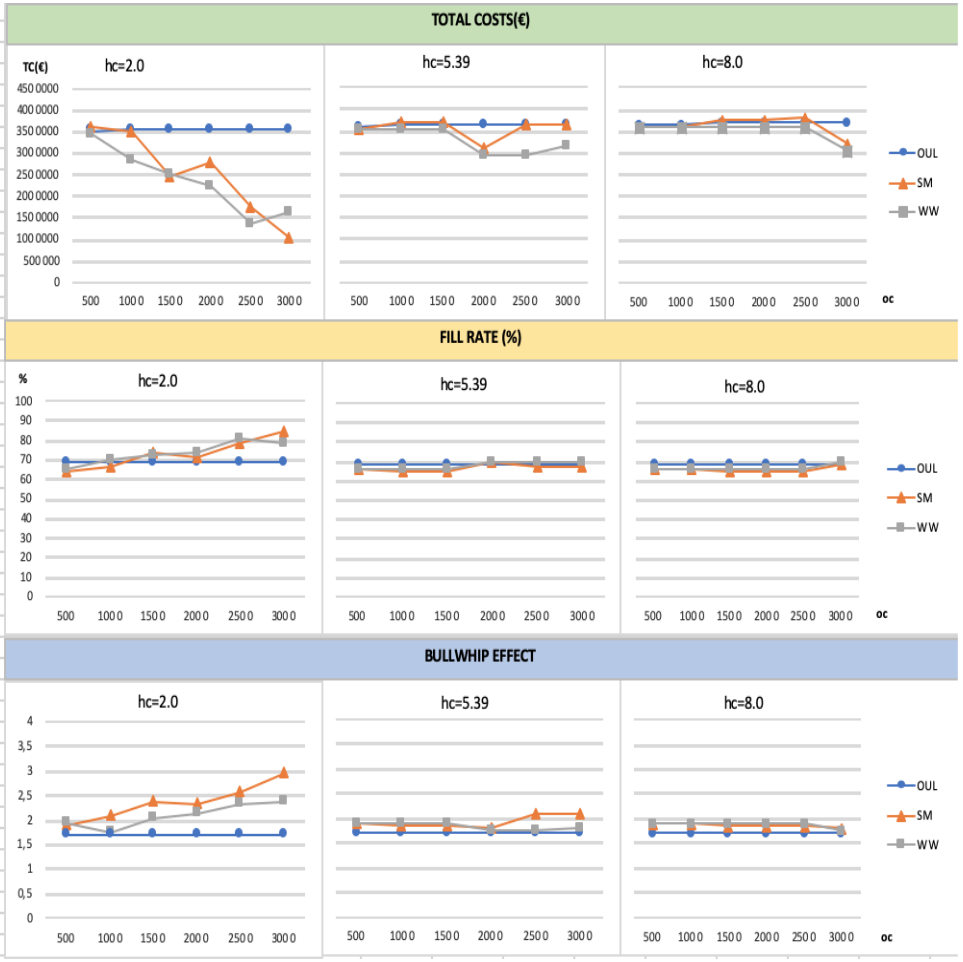
Se puede observar que se han exportado 4 hojas, una para cada tiempo de suministro, y otra en la que se hace variar el tiempo de suministro entre 2, 3 y 4 alternativamente.

Para cada hoja están representadas todas las combinaciones posibles de entradas, así como sus respuestas a los tres ‘parámetros objetivo (TC, FR y BE).

Figura 19 Libro Excel habilitado para la representación gráfica de los resultados

		CE=2.0	CE=2.0	CE=2.0	CE=5.39	CE=5.39	CE=5.39	CE=8.0	CE=8.0	CE=8.0
		Total Cost	Fill Rate	BE	Total Cost	Fill Rate	BE	Total Cost	Fill Rate	BE
500	OUL	3533758	68,85	1,7	3608703	68,85	1,7	3666402	68,85	1,7
500	SM	3602702	64,13	1,87	3530448	65,52	1,92	3591767	65,52	1,92
500	WW	3450803	65,52	1,92	3530448	65,52	1,92	3591767	65,52	1,92
1000	OUL	3542258	68,85	1,7	3617203	68,85	1,7	3674902	68,85	1,7
1000	SM	3519786	66,71	2,09	3700777	64,13	1,87	3598767	65,52	1,92
1000	WW	2846735	70,04	1,76	3537448	65,52	1,92	3598767	65,52	1,92
1500	OUL	3550758	68,85	1,7	3625703	68,85	1,7	3683402	68,85	1,7
1500	SM	2448966	73,22	2,38	3707777	64,13	1,87	3777898	64,13	1,87
1500	WW	2524407	71,96	2,05	3544448	65,52	1,92	3605767	65,52	1,92
2000	OUL	3559258	68,85	1,7	3634203	68,85	1,7	3691902	68,85	1,7
2000	SM	2798205	71,61	2,3	3118475	68,96	1,79	3784898	64,13	1,87
2000	WW	2258891	73,93	2,11	2947305	70,04	1,76	3612767	65,52	1,92
2500	OUL	3567758	68,85	1,7	3642703	68,85	1,7	3700402	68,85	1,7
2500	SM	1761450	77,99	2,58	3619195	66,71	2,09	3791898	64,13	1,87
2500	WW	1350689	81,28	2,3	2954305	70,04	1,76	3619767	65,52	1,92
3000	OUL	3576258	68,85	1,7	3651203	68,85	1,7	3708902	68,85	1,7
3000	SM	1037647	84,96	2,96	3625695	66,71	2,09	3193781	68,96	1,79
3000	WW	1624362	78,6	2,39	3132475	68,96	1,79	3027957	70,04	1,76

COPIAR Y PEGAR TABLA



6.1.5.3 - Interpretación de los resultados

Una vez obtenidos los resultados del análisis, llega la que sea quizás la parte más interesante, analizarlos. Como se comentaba anteriormente, este Ejemplo tiene la doble intención de servir como herramienta de negocio y de ser de utilidad académica.

Tomando como base del análisis las figuras 21, 22, 23 y 24, que representan el comportamiento del sistema para 4 tiempos de suministro, se va a proceder a realizar la interpretación desde los dos puntos de vista mencionados.

Figura 20 Resultados simulación para Tiempo de suministro=2 periodos

Figura 21 Resultados simulación para Tiempo de suministro=3 periodos

Figura 22 Resultados simulación para Tiempo de suministro=4 periodos

Figura 23 Resultados simulación para Tiempo de suministro variable [2,3,4] Aleatorio

Toma de decisiones

Para realizar el análisis, supongamos que ocupamos el lugar del comerciante en la cadena de suministro dada de dos niveles. Tenemos que satisfacer una demanda en los próximos 20 periodos (meses, por ejemplo), y tendremos que decidir como configurar nuestra cadena para optimizarla y cubrir los siguientes puntos:

1-Satisfacer toda la demanda (a tiempo o no, en este ejemplo).
2-Cubrir la demanda maximizando los beneficios.

Para el primer objetivo, debemos hacer que los pedidos al proveedor, lleguen a tiempo cuando los necesitemos, y para ello podemos negociar **Tiempos de Suministro** adecuados para nuestro proceso.

Para la segunda, teniendo en cuenta que la función de Costes Totales del aprovisionamiento sería:

$$CT=CT(Posesión)+CT(Emisión)+CT(adquisición)$$

El precio de compra o coste de adquisición, supongamos que es el de un mercado competitivo, es decir, sin posibilidad de negociación y fijo. Si bien es cierto, que en la realidad no sería así ya que con una cantidad mayor de pedidos se suelen obtener precios unitarios menores y sería susceptible de agregarlo a la función a optimizar, en este ejemplo por simplicidad no se tendrá en cuenta y no se minimizará.

Los costes de posesión(Ch), sí será posible decidir sobre ellos, ya que se podrá elegir almacenarlo en un sitio u otro.

Los costes de emisión(Ce), derivados de los gastos de gestión del pedido, supongamos que también tenemos opciones de optimizarlos.

Sabiendo entonces cuales son los objetivos, y teniendo información de previsión de la demanda. Se ha generado el modelo y se ha generado también una demanda real ficticia y se han obtenido las gráficas anteriores.

Las conclusiones a la vista de las gráficas, y fijándonos en los parámetros de salida **Fill Rate**(nivel de servicio) y **Total Cost** (Coste total):

Los **tiempos de suministro influyen directamente en los costes totales**, independientemente de cuál sea el coste de posesión, se aprecia claramente que los niveles obtenidos de coste total aumentan conforme aumenta el Tiempo de suministro. Véase la diferencia que existe entre TS=2 y el resto, independientemente de la lotificación utilizada.

A partir de TS=2, las diferencias en coste **vienen marcadas por el tipo de lotificación hecha.** Luego, si por ejemplo, estuviésemos obligados a pedir con TS>2, haciendo una lotificación SM o WW, obtendríamos resultados favorables.

Se observa claramente, **que la aplicación de técnicas de lotificación, mejora considerablemente los resultados en cuanto a los 2 parámetros,** Coste total y Nivel de servicio. Afinando un poco más, se puede ver que la lotificación mejora cuando los Costes de Emisión son mayores, mejorando con Tiempos de suministro menores.

Todo lo anterior, puede llevar al analista a seleccionar unas estrategias a favor de otras, así como a conocer, que parámetros de su Cadena de Suministro, son las que afectan más directamente a la generación de sus Costes Totales y su nivel de servicio dado.

Aportación académica

En este sentido, además de lo analizado anteriormente, podríamos analizar la influencia de los parámetros de entrada dados en el denominado efecto Bullwhip o efecto látigo (que estudia la variabilidad y desajustes de la demanda real en los distintos niveles de la cadena).

Un efecto BullWhip con valores superiores a 1, indican demandas desajustadas en los distintos niveles, siendo mayor este desajuste, cuanto más grande es este valor.

Con los resultados de las gráficas, se puede observar, que conforme los Ce son mayores, el efecto BE aumenta, así como con valores superiores de Ch, este valor disminuye.

La aplicación de técnicas de lotificación hace aumentar este valor, y se hace más intenso sumándose a Cotes de emisión superiores.

Con Tiempos de suministro variables, en la última gráfica, los valores de BE aumentan ligeramente respecto a los casos donde es fijo.

7-Conclusiones

Mediante la elaboración del trabajo, se ha pretendido facilitar la labor de simulación de sistemas, y concretamente de sistemas logísticos mediante la elaboración de una aplicación multi software, capaz de llevar a cabo el desarrollo del horizonte rodante combinado con la simulación de modelos. Las ventajas de dicha aplicación, derivan principalmente de que se trata de un sistema compacto que agrupa herramientas distintas y las aúna en una única estructura.

Se puede apreciar, que una vez el sistema está construido, la sencillez de trabajar con la aplicación hace que el usuario de la misma, se pueda centrar enteramente en el proceso, ya sea con fines económicos o didácticos.

Sin embargo, y dado que la aplicación utiliza muchos programas, el elaborarla quizás requiera de cierta experiencia o especialización en los softwares utilizados, así como de los conocimientos propios del sistema que se analiza. Por tanto, un inconveniente claro en el desarrollo de la herramienta, es que el personal desarrollador, debe tener una cualificación alta en materias un tanto multidisciplinares, por lo que el coste del análisis aumenta.

Una de las posibles vías que harían factible la implementación de este tipo de programas, sería el estandarizar una aplicación de modo que fuese comercializable y adaptable a distintos tipos de empresas, buscando la especificación del programa en un área concreta, por ejemplo, en la cadena de suministro. De este modo, el usuario final, adquiría el programa, y sería un consultor externo quien daría soporte para adaptarlo a sus necesidades, ciñéndose el usuario a la interpretación de los datos.

Finalmente, se puede concluir que es una buena herramienta, útil en los procesos de toma de decisiones, así como en una labor de enseñanza, pero el desarrollo de la misma, al ser muy específico, requiere de un grado de conocimiento de los Software elevados, y por tanto hace que la labor de análisis se ralentice y se encarezca.

8-Bibliografía

Definiciones dinámica de sistemas:
http://dinamicasistematicatgs.blogspot.com/2009/11/definicion-dinamica-de-sistemas_22.html

Manuales Vensim: http://www.dinamica-de-sistemas.com/vensim/vensim_1.pdf

Artículo: "*A rolling horizon simulation approach for managing demand with lead times variability"* por Francisco Campuzano-Bolarín, Josefa Mula, Manuel Díaz-Madroñero, Álvar-Ginés Legaz-Aparicio

Apuntes de la asignatura "Simulación de Modelos Logísticos" del Master en Organización Industrial (UPCT), por Francisco Campuzano-Nolarín

Conceptos básicos Java Eclipse
https://www.ibm.com/developerworks/ssa/java/tutorials/j-introtojava1/index.html

Efecto BullWhip:
http://www.atoxgrupo.com/website/noticias/efecto-latigo#targetText=El%20%22efecto%20l%C3%A1tigo%22%20o%20%22,de%20los%20centros%20de%20distribuci%C3%B3n.

Anexos

Anexo I: Algoritmos de lotificación WW y SM con Java

Wagner-Whitin

El modelo matemático de Wagner-Within(WW) (1958) está basado en programación dinámica dada por la expression:

$$F(t) = min\left[\begin{matrix}\min\limits_{1\leq j<t}\left[s_j + \sum\limits_{h=j}^{t-1}\sum\limits_{k=h+1}^{t} i_h d_k + F(j-1)\right] \\ s_t + F(t-1)\end{matrix}\right]$$

donde,

- t es el periodo de tiempo actual.
- d_t = la suma de la demanda hasta el periodo t.
- i_t = son los intereses por mantener una unidad en almacén un periodo más.
- s_t = costes de arranque en el periodo t.
- x_t = suma de las ordenes o producción en el periodo t.
- $F(t)$ = Función que minimiza los costes desde el periodo 1 hasta t.

Modelo de Silver-Meal

La técnica heurística de Silver-Meal(SM) (1977) está basada en la programación dinámica dada por la ecuación:

$$\gamma_{i,j} = \frac{1}{j-i+1}\left[k + \sum_{k=1}^{j} h(k-i)d_k\right]$$

donde,

- $\gamma_{i,j}$ es el coste por periodo si comprásemos desde el periodo i hasta el j.
- h coste de mantenimiento.

```
package silvermeal;

// SilvermealCalculator is used to calculate the lot-sizing using the heuristic technique of Silver-Meal algorithm, that is based on dynamic programming

public class SilverMealCalculator {
	private Integer [] originalDemands;		// demands
	private double orderCost;			// oc
	private double stockPerUnitPerTimeCost;		// hc

	public SilverMealCalculator(Integer [] demands, double orderCost, double stockPerUnitPerTimeCost) {
		this.originalDemands = demands;
		this.orderCost = orderCost;
		this.stockPerUnitPerTimeCost = stockPerUnitPerTimeCost;
	}

	private double calculateMeanCost(ArrayList<Integer> demands, int numberOfPeriods) {
		double result = orderCost;
		for (int step = 1; step < numberOfPeriods; step++) {
			result = result + step * demands.get(step) * this.stockPerUnitPerTimeCost;
		}
		result = result / numberOfPeriods;
		return result;
	}

	private int getNumberOfPeriods(ArrayList<Integer> demands) {
		double lastCost = Double.POSITIVE_INFINITY;
		double currentCost = 0;
		int result = 1;
		for (int period = 1; period <= demands.size(); period++)
		{
			currentCost = calculateMeanCost(demands, period);
			if (currentCost > lastCost)
				break;
			result = period;
			lastCost = currentCost;

		}
		return result;
	}

	public int [] getOrders() {
		ArrayList<Integer> demands = new ArrayList<Integer>(Arrays.asList(originalDemands));
		int currentNumberOfPeriods;
		int [] orders = new int[originalDemands.length];
		int orderIndex = 0;
		while (demands.size() > 0) {
			currentNumberOfPeriods = getNumberOfPeriods(demands);
			int accumulatedDemand = 0;
			for (int index = 0 ; index < currentNumberOfPeriods; index++) {
				accumulatedDemand = accumulatedDemand + demands.get(0);
				demands.remove(0);
```

Figura 24 Código en Java de algoritmo Silver-Meal

```
				orderIndex = orderIndex + currentNumberOfPeriods;

			}
			return orders;
		}
```

```
package wagnerwhitin;

// WagnerWithinCalculator is used to calculate the optimal lot-sizing using the Wagner-Whitin algorithm that is based on
dynamic programming

public class WagnerWhitinCalculator {

          private float [] originalDemands;             // demands
          private double orderCost;                     // oc
          private double stockPerUnitPerPeriodCost;     // hc

          public WagnerWhitinCalculator(float [] demands, double orderCost, double stockPerUnitPerPeriodCost) {
                    this.originalDemands = demands;
                    this.orderCost = orderCost;
                    this.stockPerUnitPerPeriodCost = stockPerUnitPerPeriodCost;
          }

          public double getCost(int fromPeriod, int toPeriod) {
                    double cost = orderCost;

                    for (int period=fromPeriod + 1; period <= toPeriod; period++) {
                              int step = period - fromPeriod;
                              cost = cost + step * originalDemands[period] * stockPerUnitPerPeriodCost;
                    }
                    return cost;
          }

          public float [] getOrders() {
                    double [] costTable = new double[originalDemands.length + 1];
                    int [] fromTable = new int[originalDemands.length + 1];
                    costTable[0] = 0.0;

                    for (int numberOfPeriods=1; numberOfPeriods < costTable.length; numberOfPeriods++) {
                              costTable[numberOfPeriods] = Double.POSITIVE_INFINITY;

                              for     (int     fromNumberOfPeriods=0;     fromNumberOfPeriods     <     numberOfPeriods;
fromNumberOfPeriods++) {

                                        double cost = costTable[fromNumberOfPeriods] + getCost(fromNumberOfPeriods,
numberOfPeriods - 1);
                                        if (cost <= costTable[numberOfPeriods])
                                                  costTable[numberOfPeriods] = cost;
                                                  fromTable[numberOfPeriods] = fromNumberOfPeriods;
                                        }
                              }
                    }
                    float [] orders = new float [originalDemands.length];
                    int index = originalDemands.length;
                    int ordersIndex = originalDemands.length - 1;

                    while (index != 0) {
                              int currentNumberOfPeriods = index - fromTable[index];
                              int accumulatedOrders = 0;
                              for (int demandsIndex=ordersIndex; demandsIndex > ordersIndex - currentNumberOfPeriods;
demandsIndex--)
                                        accumulatedOrders += originalDemands[demandsIndex];
                              ordersIndex -= currentNumberOfPeriods;
                              orders[ordersIndex + 1] = accumulatedOrders;
                              index = fromTable[index];
                    }
                    return orders;    // optimum orders
          }


}
```

Figura 25 Código en Java de algoritmo Wagner Within

Anexo II: Tutorial básico VenSim/Java/Excel

Manual Básico para controlar Vensim en Java usando Excel

En este tutorial se expondrán los pasos a seguir para instalar y ejecutar Vensim en Java usando también Excel.

Clases necesarias

En primer lugar vamos a describir las clases necesarias para la correcta ejecución de Vensim en java.

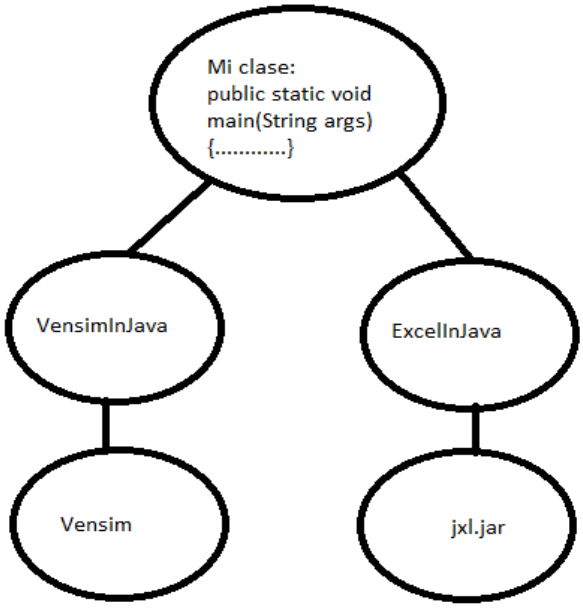
Clase Vensim.java que es la que proporciona Vensim y se utiliza para manejar las llamadas a Vensim. Esta clase ha sido implementada por los desarrolladores de Vensim y es necesaria en todos los proyectos.

Clase VensimInJava.java es de implementación propia y su cometido es realizar las llamadas a Vensim de una forma más amigable. Simplificando las funciones que proporciona VenSim, ya que estas son numerosas y no necesarias todas ellas. Es una simplificación.

Clase ExcelInVensim.java es una clase de implementación propia que llama de forma amigable a la librería jxl.jar. Al igual que la anterior, es una clase que simplifica las funciones utilizadas para el control del Excel en Java, desarrollando solo aquellas que vayan a ser utilizadas de forma frecuente.

Librería jxl.jar es la que proporciona los métodos a la clase ExcelInJava para poder manejar los archivos Excel.

Mi_clase: Es el Main del programa a partir del cual se estructura todo el programa. Todas las llamadas a las clases de VenSim y Java se harán desde aquí. Como se puede ver contiene el Main del programa y será la función que ejecute el programa principal.



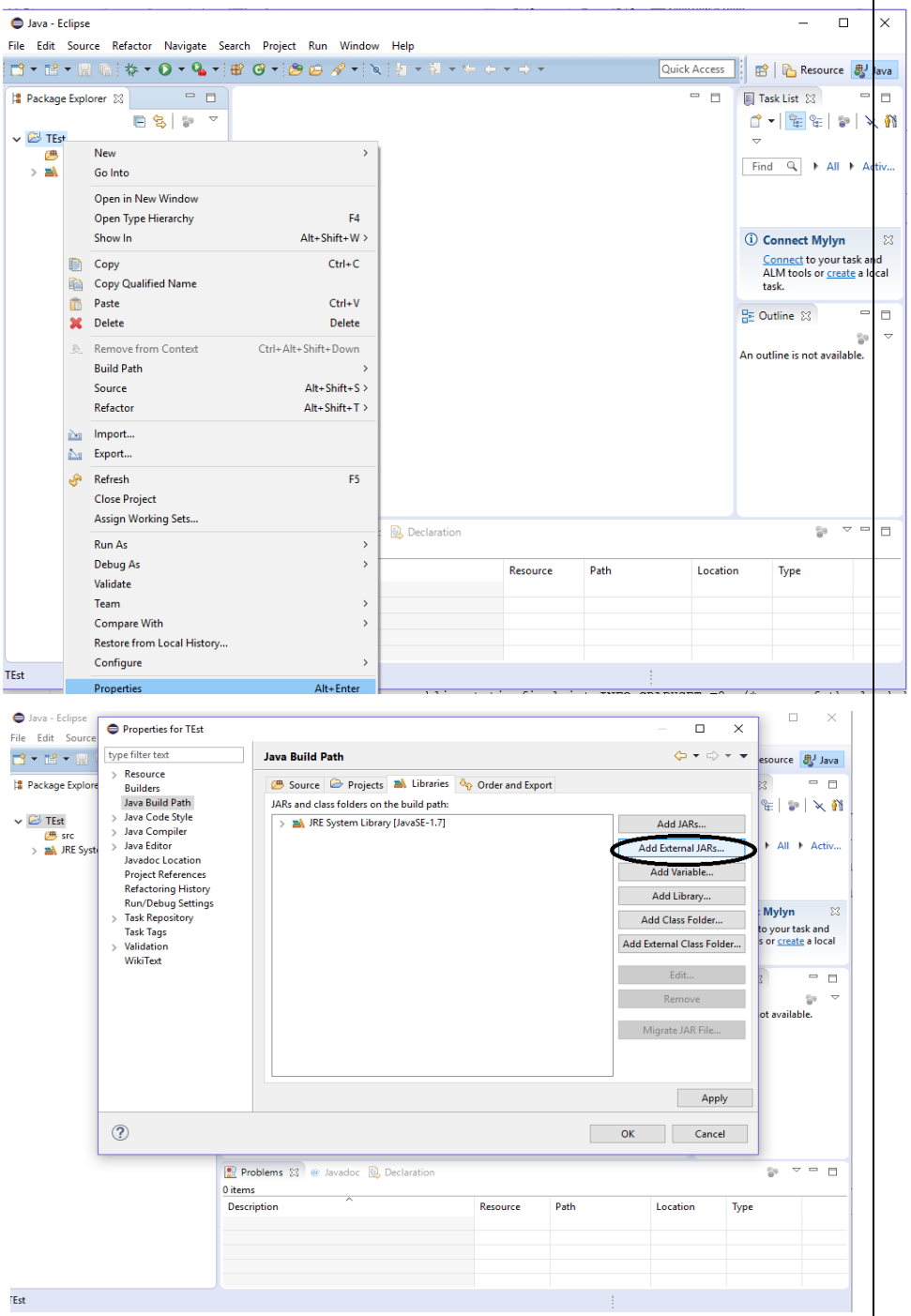
Instalación

Para instalar correctamente todas las clases es necesario realizar los siguientes pasos, en el orden establecido.

1. Descargar JDK 1.7 o superior (32bits) e instalarlo.
2. Descargar Eclipse versión 32 bits ejecutar instalador e instala Java IDE for Java developers o cualquier versión compatible.
3. Crear mi proyecto java.
    a. Crear Proyecto: Ir a File>New>Project. Después Seleccionar Java Project dentro del directorio Java.
    b. Dentro del proyecto creado (con el mouse pulsar botón derecho encima del nombre del proyecto) , crear un paquete llamado **com.vensim**. Después Crear la clase llamada **Vensim** en paquete **com.vensim** y pegar el código del Anexo1. De esta forma se estará introduciendo las librerías necesarias para el manejo de objetos VenSim desde Java.
    c. Crear un paquete llamado **com.miVensim** que contenga la clase VensimInJava. Pega(sobrescribe) en esta nueva clase el contenido del Punto A.
    d. Crear un paquete llamado **com.miExcel** que contenga la clase **ExcelInVensim** Pega(sobrescribe) en esta nueva clase el contenido del Punto C.
    e. Crear un paquete que contenga mi clase, por ejemplo com.HorizontesRodantes.

f. Buscar, Descargar e Importar la librería **jxl.jar**

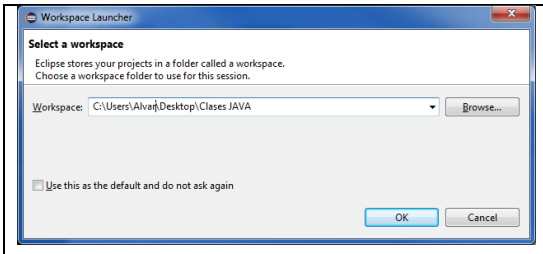
Pasos para importar librería jxl.jar

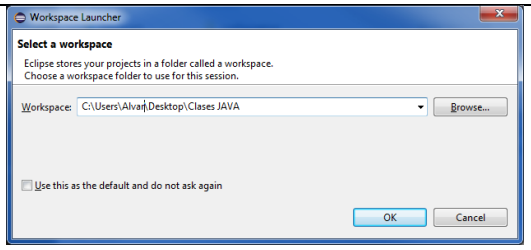


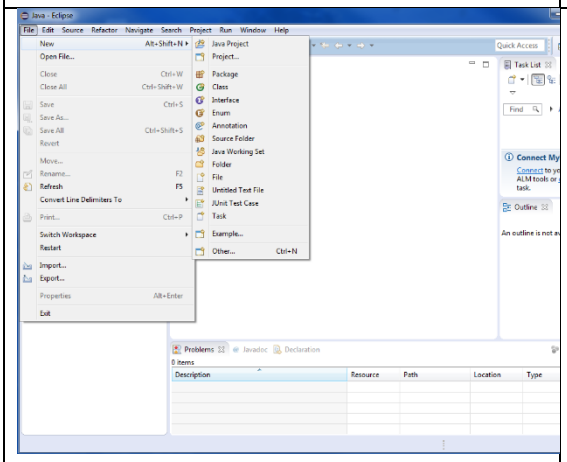
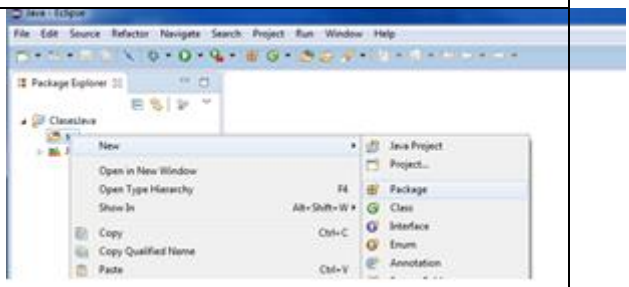
Y seleccionas el archivo **jxl.jar**

g. Crear el paquete donde tengas las clases de tu programa que llamen a los paquetes anteriores.

A continuación se muestras capturas de pantalla del proceso de creación de paquetes y clases:

<table>
<tr>
<td> 

1 Abra ECLIPSE y seleccione el espacio de trabajo, el directorio donde creará su proyecto JAVA e incluirá las clases de java.</td>
<td> 

2 Cree un nuevo proyecto JAVA</td>
</tr>
<tr>
<td> 
3 Cree un nuevo paquete, este contendrá las diferentes clases de JAVA.</td>
<td> 
4 Seleccione paquete y genere uno nuevo o seleccione clase y genere una nueva.</td>
</tr>
</table>

Manejo de Clases

Clase **ExcelInVensim.java**

Creamos un objeto de la clase ExcelInVensim con la dirección del archivo que vamos a leer y del archivo en el que vamos a escribir:

ExcelInVensim mi_excel = new ExcelInVensim("C:\\Users\\Paco\\Desktop\\INVESTIGACION_JAVA\\INVESTIGACION JAVA\\Modelo_Horizonte\\demandas.xls", "C:\\Users\\Paco\\Desktop\\INVESTIGACION_JAVA\\INVESTIGACION JAVA\\Modelo_Horizonte\\ordenes.xls");

Lectura de datos desde Excel

int hoja = 0;
int columna = 0;
int inicio = 1;
int fin= 10;
float[] datos =mi_excel.get_xls_data(hoja,columna, inicio, fin);

Escritura de datos a Excel

Primero definimos la hoja donde vamos a escribir con la posición y el nombre:

mi_excel.create_sheet("ORDERS", 0);

Ahora introducimos los datos a escribir, se puede realizar la escritura dato a dato o por vector:

Dato a dato

mi_excel.save_xls_data(hoja, columna, fila, dato);

Vector

mi_excel.save_xls_data_vector(hoja, column, from, vector);

Una vez escrito los datos, estos no se guardarán en el fichero de escritura hasta que cerremos la clase para ello se llama al siguiente método mi_excel.close_book_actualizado(); // Cierra el libro de escritura
mi_excel.close_book(); // Cierra el libro de lectura

EJEMPLO

```
package ejemplos;

import com.miExcel.ExcelInVensim;

public class ejemplo_lectura_escritura_excel {
public static void main(String args[] ){

      String
direccion_lectura="C:\\Users\\Francisco\\Desktop\\ordenes.xls";
      String
direccion_escritura="C:\\Users\\Francisco\\Desktop\\ordenesmodificadas
.xls";
      ExcelInVensim             mi_excel             =             new
ExcelInVensim(direccion_lectura,direccion_escritura);
```

```
		int hoja = 0;
		int columna = 1;
		int inicio = 0;
		int fin = 10;
		float[] datos =mi_excel.get_xls_data(hoja,columna, inicio, fin);

		System.out.println("datos leidos");

		for (int i=0;i<datos.length;i++){
			System.out.println("datos= "+datos[i]);
		}

		float [] datos_modificados=new float[datos.length];

		mi_excel.create_sheet("NUEVOS DATOS", 0);

		for (int i=0;i<datos.length;i++){
			datos_modificados[i]=datos[i]*5;
			mi_excel.save_xls_data(0, 0, i, i+1);//GRABO NUMERO A NÚMERO
		}



		mi_excel.save_xls_data_vector(0,   1,   0,   datos_modificados);//
GRABO EL VECTOR COMPLETO


		mi_excel.close_book_actualizado(); // Cierra el libro de escritura
		mi_excel.close_book(); // Cierra el libro de lectura

}
}
```

A- Librería VenSim en Java

```
/**
* Vensim class for defining the methods in the venjava.dll
* which in turn calls vensim.dll
*/
package com.vensim;
/* NOTE if you change the package name you will need to rebuild
   venjava.dll using the new javah generated function declarations */

//import java.awt.*;

public class Vensim
{
/* variable types  for get_varnames */
public static final int VARTYPE_WORKBENCH =-1;
public static final int VARTYPE_ALL =0;
public static final int VARTYPE_LEVEL =1;
public static final int VARTYPE_AUXILIARY =2;
public static final int VARTYPE_DATA =3;
public static final int VARTYPE_INIITIAL =4;
public static final int VARTYPE_CONSTANT =5;
public static final int VARTYPE_LOOKUP =6;
public static final int VARTYPE_GROUP =7;
public static final int VARTYPE_SUBSCRIPT =8;
public static final int VARTYPE_CONSTRAINT =9;
public static final int VARTYPE_TEST_INPUT =10;
public static final int VARTYPE_TIME_BASE =11;
public static final int VARTYPE_GAME =12;
public static final int VVARTYPE_SUBSCRIPT_CONSTANT =13;


/* attribute types for get_varattrib */
public static final int ATTRIB_UNITS =1;
public static final int ATTRIB_COMMENT =2;
public static final int ATTRIB_EQUATIONS =3;
public static final int ATTRIB_CAUSES =4;
public static final int ATTRIB_USES =5;
public static final int ATTRIB_INITCAUSES =6; /* outputs only initial causes */
public static final int ATTRIB_ACTIVECAUSES =7; /* outputs only active causes - not initial */
public static final int ATTRIB_SUBFAMILY =8; /* list the subscript ranges associated with the variable */
public static final int ATTRIB_SUBALL =9; /* lists the expanded subscript list for the variable */
public static final int ATTRIB_SUBWORK =10; /* lists the expanded set of subscipts that would be used on tool invocation */
public static final int ATTRIB_MIN =11;
public static final int ATTRIB_MAX =12;
public static final int ATTRIB_INCREMENT =13;
public static final int ATTRIB_VARTYPE =14;
public static final int ATTRIB_GROUP =15;

/* return values for check_status */
public static final int STATUS_IDLE =0;
public static final int STATUS_SIMULATING =1;
public static final int STATUS_SIMHANG =2;
public static final int STATUS_BLOCKACTION =3;
public static final int STATUS_MEMLOCK =4;
public static final int STATUS_INGAME =5;
public static final int STATUS_NEEDFREE =6;
```

```
/* information queries for get_info */
public static final int INFO_DLL =1;/* returns Minimal, Silent, Full or Redist */
public static final int INFO_VERSION =2; /* version info for Vensim */
public static final int INFO_USER =3;/* user name \0 user company \0 */
public static final int INFO_DIRECTORY =4 ;/* the currently active directory */
public static final int INFO_MODELNAME =5 ;/* the name of the currently loaded model (no directory) */
public static final int INFO_TOOLSET =6 ;/* the name of the toolset */
public static final int INFO_TOOLLIST =7 ;/* the names of the tools in the loaded toolset */
public static final int INFO_GRAPHSET =8 ;/* name of the loaded graph set */
public static final int INFO_GRAPHLIST =9 ;/* list of available graphs */
public static final int INFO_RUNLIST =10 ;/* the list of loaded runs */
public static final int INFO_RUNNAME =11 ;/* the name of the run to be maded changed with SIMULATE>RUNNAME */
public static final int INFO_CINFILES =12 ;/* cin files set with SIMULATE>CHGFILE */
public static final int INFO_DATAFILES =13 ;/* data file set with SIMULATE>DATA */
public static final int INFO_BASED =14 ;/* name of run to base on set with SIMULATE>BASED */
public static final int INFO_OPTMARM =15 ;/* optimization parameter files SIMULATE>OPTPARM */
public static final int INFO_PAYOFF =16 ;/* name of payoff control files SIMULATE>PAYOFF */
public static final int INFO_RESUME =17 ;/* resume satus (0 or 1) SIMULATE>RESUME */
public static final int INFO_SAVELIST =18 ;/* name of savelist file SIMULATE>SAVELIST */
public static final int INFO_SENSSAVELIST =19 ;/* name of sensitivity save list files SIMULATE>SENSSAVELIST */
public static final int INFO_SENSITIVITY =20 ;/* name of sensitity contol file SIMULATE>SENSITIVITY */
public static final int INFO_BENCHVAR =21 ;/* workbench var name as it would appear in title bar */
public static final int INFO_VIEWLIST =22 ; /* list of views in the model */
public static final int INFO_TIMEAXIS =23 ; /* min\0max\0special */


/* vensim be quiet flags*/
public static final int QUIET_NORMAL=0;
public static final int QUIET_NO_WIP=1;
public static final int QUIET_NO_WIP_NO_DIALOGS=2;

        /**
        * Native method declarations
        */
        static native int get_jversion() ;
        public static native int command( String command );
        public static native int CCommand(int context,String command );
        public static native int get_data( String fileName, String varName, String timeAxisName, float vectorOfValues[], float
timeVectorValues[], int maxNumberValues );
        public static native int CGetData(int context, String fileName, String varName, String timeAxisName, float
vectorOfValues[], float timeVectorValues[], int maxNumberValues );
        public static native int tool_command( String command, int windowHandle, int aswiptool );//generally references to
windows just won't work
        public static native int CToolCommand(int context, String command, int windowHandle, int aswiptool );//generally
references to windows just won't work
        public static native int start_simulation(int loadFirst, int game, int overwrite );
        public static native int CStartSimulation(int context, int loadFirst, int game, int overwrite );
        public static native int continue_simulation( int numIntervals );
        public static native int CContinueSimulation(int context, int numIntervals );
        public static native int finish_simulation();
        public static native int CFinishSimulation(int context);
        public static native int get_val( String varName, float val[] );
        public static native int CGetVal(int context, String varName, float val[] );
        public static native int get_dpval( String varName, double val[] );
        public static native int CGetDPVval(int context, String varName, double val[] );
        public static native int show_sketch( int sketchNum, int wantScroll, int zoomPercent, int windowHandle );//generally
references to windows just won't work
        public static native int CShowSketch(int context, int sketchNum, int wantScroll, int zoomPercent, int windowHandle
);//generally references to windows just won't work
        public static native int be_quiet( int quietflag );
        public static native int CBeQuiet(int context, int quietflag );
        public static native int check_status();
        public static native int CCheckStatus(int context);
        public static native int ContextAdd(int wantcleanup) ;
        public static native int ContextDrop(int context) ;

        /* these methods return string arrays - if there is no information found
           the lenght of these arrays is 0 */
        public static native String[] get_varnames(String filter, int vartype) ;
```

```
        public static native String[] CGetVarNames(int context,String filter, int vartype) ;
        public static native String[] get_varattrib(String varname,int attrib) ;
        public static native String[] CGetVarAttrib(int context,String varname,int attrib) ;
        public static native String[] get_info(int infowanted) ;
        public static native String[] CGetInfo(int context,int infowanted) ;
        /* note the get_info almost always returns an array of length 1 */

        /* this method is of no use and has been commented out but is
          still supported by venjava.dll
          public static native int get_substring(String fullstring,int offset,String buf,int maxbuflen) ;
          */

  public static native int get_varoff(String varname) ;
  public static native int CGetVarOff(int context,String varname) ;
        public static native int get_vecvals(int vecoff[],float val[],int nvals)  ;
        public static native int CGetVecVals(int context,int vecoff[],float val[],int nvals)  ;
        public static native int get_dpvecvals(int offsets[],double dpvals[],int veclen) ;
        public static native int CGetDPVecVals(int context,int offsets[],double dpvals[],int veclen) ;
        public static native int get_sens_at_time(String filename,String varname,String tname,float intime[],float vals[],int maxn)
;
        public static native int CGetSensAtTime(int context,String filename,String varname,String tname,float intime[],float
vals[],int maxn) ;
        public static native int set_parent_window(int window,int r1,int r2) ;//generally references to windows just won't work
        public static native int CSetParentWindow(int context,int window,int r1,int r2) ;//generally references to windows just
won't work


        /**
        * Static, load the dynamic library
        */
        public Vensim(String libname)
        {
    try
    {

//      System.out.println("Library path is " + System.getProperty("java.library.path","")) ;
        System.loadLibrary(libname);
                                int rval = Vensim.get_jversion() ;
                                if(rval != 5050) {
                                  System.out.println("Bad version " + rval + " java class does not match DLL") ;
                                  }
            }
    catch( UnsatisfiedLinkError e )
    {
      System.out.println( "Cannot load native library. Error: " + e.toString() );
    }
        }


}
```

B - Clase VenSim

```
package com.miVensim;

import com.vensim.Vensim;

public class VensimInJava {

	private Vensim vensim; // con vensim.command controlamos a vensim

	// Building method (Load dll)
	public VensimInJava()
	{
		vensim = new Vensim("vendll32"); /* vendml32 for the minimal dll */
	}
	// Load model

	public void loadModel(String path)
	{

	   int result = Vensim.command( "SPECIAL>LOADMODEL|"+path );
		if (result == 0)
			System.out.println( "Model -- failed" );
		else if (result == 1)
		{
			System.out.println( "Vensim loaded model" );

		}

	}

	// Create simulation

	public void createSimulation(String simulationName){

		int result = Vensim.command( "SIMULATE>RUNNAME|"+simulationName);
		if (result == 0)
		System.out.println( "SIMULATE>RUNNAME|base -- failed" );

	}
	// Run simulation

	public void runSimulation(){

		int result = Vensim.command( "MENU>RUN|O" );
		if (result == 0)

			System.out.println( "MENU>RUN|O -- failed" );
	}

	// Create GAMEON

	public void createGameon(){

		int result = Vensim.command( "GAME>GAMEINTERVAL|1" ); // Se define el intervalo
		if (result == 0)
			System.out.println("set game interval fails");
		else
			System.out.println("set game interval succeeds");

		//result = Vensim.command("SIMULATE>SETVAL|aleatorios ="+String.valueOf(datos[0]));
		result = Vensim.command("MENU>GAME"); // Se arranca el GAME
		if (result == 0)
			System.out.println("Start game fails");
		else
			System.out.println("Start game succeeds");

	}
```

```
// Set GAME VAR poner un valor determinado a una VARIABLE GAMING

public void setVar(String VarName, float value){

        int result = vensim.command("SIMULATE>SETVAL|"+VarName+" = "+String.valueOf(value));

}


// Get GAME VAR

public float getVar(String VarName){

        float[] var = new float[500];
        int nResults = vensim.get_val(VarName, var);

        return var[0];

}



// Run GAMEON
public void gameOn(){

        int result = Vensim.command("GAME>GAMEON");
        if (result == 1)
                System.out.println( "game in progress" );
        else
                System.out.println( "gameon -- failed" );
}
// Close GAMEON
public void endGame(){



}

}
```

C-Clases Excel

```
package com.miExcel;

import java.io.File;
import java.io.IOException;
import java.util.Vector;

import jxl.Sheet;
import jxl.Workbook;
import jxl.read.biff.BiffException;
import jxl.write.WritableSheet;
import jxl.write.WritableWorkbook;
import jxl.write.WriteException;


/*
Clase para leer y guardar archivos xls. Utiliza como soporte la librería de código abierto JXL.
*/
public class ExcelInVensim {

	private Workbook libro_entrada;
	private WritableWorkbook libro_salida;
	private Vector<WritableSheet> hojas;


	public ExcelInVensim (String direccion_archivoExcel, String dir_libro_salida){

		try {

			libro_entrada = Workbook.getWorkbook(new File(direccion_archivoExcel));
			//Workbook target_workbook = Workbook.getWorkbook(new File(dir_libro_salida));
	libro_salida = Workbook.createWorkbook(new File(dir_libro_salida));
	//target_workbook.close();

		} catch (BiffException | IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		hojas = new Vector<WritableSheet>();

	}

	public  float[] get_xls_data(int num_hoja,int columna,int from,int to)
	{

		float[] salida = new float[to-from+1];
			try {

			Sheet hoja = libro_entrada.getSheet(num_hoja);


			int numColumnas = hoja.getColumns();
			int numFilas = hoja.getRows();

			// Se comprueba que el número de columnas coincide
			if(columna<numColumnas & to<numFilas){

				String data;
				// Recorre cada fila de la hoja
				for (int fila = from; fila <= to; fila++) {

					// Solo se muestra una columna determinada de la hoja de datos
					data = hoja.getCell(columna, fila).getContents();
					salida[fila-from]=Float.valueOf(formatoPunto(data));
				}
			}// Final del if
```

```
                        } // Final del TRY
                        catch (Exception ioe) {
                        ioe.printStackTrace();
                        }

                return salida;

                }

public void create_sheet(String sheet_name,int pos){

                WritableSheet hoja = libro_salida.createSheet(sheet_name,pos);
                hojas.add(hoja);
}

public void  save_xls_data_vector( int hoja, int column, int from,float[] value)
{
                        int num_filas = value.length;
        try{

           // Se escriben los valores en las celdas
           for(int fila=0+from; fila<num_filas+from; fila++){
                 jxl.write.Number number = new jxl.write.Number(column, fila, value[fila-from]);
                 ((WritableSheet) hojas.elementAt(hoja)).addCell(number);
           }
           //libro_salida.write();
        }
        catch(Exception e)
        {
           System.out.println("writeExcel ->"+e);
        }
}

public void  save_xls_data( int hoja, int column, int fila,float value)
{

        try{
            jxl.write.Number number = new jxl.write.Number(column, fila, value);
            ((WritableSheet) hojas.elementAt(hoja)).addCell(number);
        }
        catch(Exception e)
        {
           System.out.println("writeExcel ->"+e);
        }

}


public void close_book_actualizado(){

                try
                {
                        //libro_entrada.close();
                        libro_salida.write();
                        libro_salida.close();

                }
                catch (WriteException | IOException e)
                {
                        e.printStackTrace();
                }
}



public void close_book(){
```

```
                        libro_entrada.close();


        }

        private String formatoPunto(String datos){
                        char[] datosChar = datos.toCharArray();
                        for(int i=0;i<datosChar.length;i++){
                                if(datosChar[i]==','){
                                        datosChar[i] = '.';
                                }
                        }
                        return String.valueOf(datosChar);
        }

}
```

APLICACIÓN INFORMÁTICA PARA LA GESTIÓN DE LA DEMANDA EN CADENAS DE SUMINSITRO UTILIZANDO EL HORIZONTE RODANTE Y TÉCNICAS DE LOTIFICACIÓN